stable

Release 1.3.4

Aug 11, 2018

Contents

Pilbox is an image processing application server built on Python's Tornado web framework using the Python Imaging Library (Pillow). It is not intended to be the primary source of images, but instead acts as a proxy which requests images and resizes them as desired.

CHAPTER 1

Setup

1.1 Dependencies

- Python >= 2.7
- Pillow 5.2.0
- Tornado 5.1.0
- OpenCV 2.x (optional)
- PycURL 7.x (optional, but recommended; required for proxy requests and requests over TLS)
- Image Libraries: libjpeg-dev, libfreetype6-dev, libwebp-dev, zlib1g-dev, liblcms2-dev

Pilbox highly recommends installing `libcurl` and `pycurl` in order to get better HTTP request performance as well as additional features such as proxy requests and requests over TLS. Installed versions of `libcurl` should be a minimum of `7.21.1` and `pycurl` should be a minimum of `7.18.2`. Furthermore, it is recommended that the `libcurl` installation be built with asynchronous DNS resolver (threaded or c-ares), otherwise it may encounter various problems with request timeouts (for more information, see CURLOPT_CONNECTTIMEOUT_MS and comments in curl_httpclient.py)

1.2 Install

Pilbox can be installed with pip

```
$ pip install pilbox
```

Or easy_install

```
$ easy_install pilbox
```

Or from source

```
$ git clone https://github.com/agschwender/pilbox.git
```

CHAPTER 2

Running

To run the application, issue the following command

```
$ python -m pilbox.app
```

By default, this will run the application on port 8888 and can be accessed by visiting:

```
http://localhost:8888/
```

To see a list of all available options, run

```
$ python -m pilbox.app --help
Usage: pilbox/app.py [OPTIONS]

Options:

  --allowed_hosts            list of allowed hosts (default [])
  --allowed_operations       list of allowed operations (default [])
  --background               default hexadecimal bg color (RGB or ARGB)
  --ca_certs                 filename of CA certificates in PEM format
  --client_key               client key
  --client_name              client name
  --config                   path to configuration file
  --content_type_from_image  override content type using image mime type
  --debug                    run in debug mode
  --expand                   default to expand when rotating
  --filter                   default filter to use when resizing
  --help                     show this help information
  --implicit_base_url        prepend protocol/host to url paths
  --max_operations           maximum operations to perform (default 10)
  --max_requests             max concurrent requests (default 40)
  --max_resize_height        maximum resize height (default 15000)
  --max_resize_width         maximum resize width (default 15000)
  --operation                default operation to perform
  --optimize                 default to optimize when saving
```

(continues on next page)

(continued from previous page)

```
  --port                    run on the given port (default 8888)
  --position                default cropping position
  --preserve_exif           default behavior for Exif information
  --progressive             default to progressive when saving
  --proxy_host              proxy hostname
  --proxy_port              proxy port
  --quality                 default jpeg quality, 1-99 or keep
  --retain                  default adaptive retain percent, 1-99
  --timeout                 timeout of requests in seconds (default 10)
  --user_agent              user agent
  --validate_cert           validate certificates (default True)
  --workers                 number of worker processes (0 = auto) (default 0)
```

CHAPTER 3

Calling

To use the image processing service, include the application url as you would any other image. E.g. this image url

```
<img src="http://i.imgur.com/zZ8XmBA.jpg" />
```

Would be replaced with this image url

```
<img src="http://localhost:8888/?url=http%3A%2F%2Fi.imgur.com%2FzZ8XmBA.jpg&w=300&
↪h=300&mode=crop" />
```

This will request the image served at the supplied url and resize it to `300x300` using the `crop` mode. The below is the list of parameters that can be supplied to the service.

3.1 General Parameters

- *url*: The url of the image to be resized
- *op*: The operation to perform: noop, region, resize (default), rotate
  - *noop*: No operation is performed, image is returned as it is received
  - *region*: Select a sub-region from the image
  - *resize*: Resize the image
  - *rotate*: Rotate the image
- *fmt*: The output format to save as, defaults to the source format
  - *gif*: Save as GIF
  - *jpeg*: Save as JPEG
  - *png*: Save as PNG
  - *webp*: Save as WebP
  - *tiff*: Save as TIFF

- *bg*: Background color used with images that have transparency; useful when saving to a format that does not support transparency
  - *RGB*: 3- or 6-digit hexadecimal number
  - *ARGB*: 4- or 8-digit hexadecimal number, only relevant for PNG images
- *opt*: The output should be optimized, only relevant to JPEGs and PNGs
- *exif*: Keep original Exif data in the processed image, only relevant for JPEG
- *prog*: Enable progressive output, only relevant to JPEGs
- *q*: The quality, (1-99) or keep, used to save the image, only relevant to JPEGs

3.2 Resize Parameters

- *w*: The desired width of the image
- *h*: The desired height of the image
- *mode*: The resizing method: adapt, clip, crop (default), fill and scale
  - *adapt*: Resize using crop if the resized image retains a supplied percentage of the original image; otherwise fill
  - *clip*: Resize to fit within the desired region, keeping aspect ratio
  - *crop*: Resize so one dimension fits within region, center, cut remaining
  - *fill*: Fills the clipped space with a background color
  - *scale*: Resize to fit within the desired region, ignoring aspect ratio
- *bg*: Background color used with fill mode (RGB or ARGB)
  - *RGB*: 3- or 6-digit hexadecimal number
  - *ARGB*: 4- or 8-digit hexadecimal number, only relevant for PNG images
- *filter*: The filtering algorithm used for resizing
  - *nearest*: Fastest, but often images appear pixelated
  - *bilinear*: Faster, can produce acceptable results
  - *bicubic*: Fast, can produce acceptable results
  - *antialias*: Slower, produces the best results
- *pos*: The crop position
  - *top-left*: Crop from the top left
  - *top*: Crop from the top center
  - *top-right*: Crop from the top right
  - *left*: Crop from the center left
  - *center*: Crop from the center
  - *right*: Crop from the center right
  - *bottom-left*: Crop from the bottom left
  - *bottom*: Crop from the bottom center

  - *bottom-right*: Crop from the bottom right
  - *face*: Identify faces and crop from the midpoint of their position(s)
  - *x,y*: Custom center point position ratio, e.g. 0.0,0.75
- *retain*: The minimum percentage (1-99) of the original image that must still be visible in the resized image in order to use crop mode

3.3 Region Parameters

- *rect*: The region as x,y,w,h; x,y: top-left position, w,h: width/height of region

3.4 Rotate Parameters

- *deg*: The desired rotation angle degrees
  - *0-359*: The number of degrees to rotate (clockwise)
  - *auto*: Auto rotation based on Exif orientation, only relevant to JPEGs
- *expand*: Expand the size to include the full rotated image

3.5 Security-related Parameters

- *client*: The client name
- *sig*: The signature

The `url` parameter is always required as it dictates the image that will be manipulated. `op` is optional and defaults to `resize`. It also supports a comma separated list of operations, where each operation is applied in the order that it appears in the list. Depending on the operation, additional parameters are required. All image manipulation requests accept `exif`, `fmt`, `opt`, `prog` and `q`. `exif` is optional and default to `0` (not preserved). `fmt` is optional and defaults to the source image format. `opt` is optional and defaults to `0` (disabled). `prog` is optional and default to `0` (disabled). `q` is optional and defaults to `90`. To ensure security, all requests also support, `client` and `sig`. `client` is required only if the `client_name` is defined within the configuration file. Likewise, `sig` is required only if the `client_key` is defined within the configuration file. See the *Signing* section for details on how to generate the signature.

For resizing, either the `w` or `h` parameter is required. If only one dimension is specified, the application will determine the other dimension using the aspect ratio. `mode` is optional and defaults to `crop`. `filter` is optional and defaults to `antialias`. `bg` is optional and defaults to `0fff`. `pos` is optional and defaults to `center`. `retain` is optional and defaults to `75`.

For region sub-selection, `rect` is required. For rotating, `deg` is required. `expand` is optional and defaults to `0` (disabled). It is recommended that this feature not be used as it typically does not produce high quality images.

Note, all built-in defaults can be overridden by setting them in the configuration file. See the *Configuration* section for more details.

CHAPTER 4

Examples

The following images show the various resizing modes in action for an original image size of `640x428` that is being resized to `500x400`.

4.1 Adapt

The adaptive resize mode combines both *crop* and *fill* resize modes to ensure that the image always matches the requested size and a minimum percentage of the image is always visible. Adaptive resizing will first calculate how much of the image will be retained if crop is used. Then, if that percentage is equal to or above the requested minimum retained percentage, crop mode will be used. If it is not, fill will be used. The first figure uses a `retain` value of `80` to illustrate the adaptive crop behavior.

Whereas the second figure requires a minimum of `99` to illustrate the adaptive fill behavior

4.2 Clip

The image is resized to fit within a `500x400` box, maintaining aspect ratio and producing an image that is `500x334`. Clipping is useful when no portion of the image can be lost and it is acceptable that the image not be exactly the supplied dimensions, but merely fit within the dimensions.

4.3 Crop

The image is resized so that one dimension fits within the `500x400` box. It is then centered and the excess is cut from the image. Cropping is useful when the position of the subject is known and the image must be exactly the supplied size.

4.4 Fill

Similar to clip, fill resizes the image to fit within a `500x400` box. Once clipped, the image is centered within the box and all left over space is filled with the supplied background color. Filling is useful when no portion of the image can be lost and it must be exactly the supplied size.

4.5 Scale

The image is clipped to fit within the `500x400` box and then stretched to fill the excess space. Scaling is often not useful in production environments as it generally produces poor quality images. This mode is largely included for completeness.

CHAPTER 5

Signing

In order to secure requests so that unknown third parties cannot easily use the resize service, the application can require that requests provide a signature. To enable this feature, set the `client_key` option. The signature is a hexadecimal digest generated from the client key and the query string using the HMAC-SHA1 message authentication code (MAC) algorithm. The below python code provides an example implementation.

```
import hashlib
import hmac

def derive_signature(key, qs):
    m = hmac.new(key, None, hashlib.sha1)
    m.update(qs)
    return m.hexdigest()
```

The signature is passed to the application by appending the `sig` parameter to the query string; e.g. `x=1&y=2&z=3&sig=c9516346abf62876b6345817dba2f9a0c797ef26`. Note, the application does not include the leading question mark when verifying the supplied signature. To verify your signature implementation, see the `pilbox.signature` command described in the *Tools* section.

CHAPTER 6

Configuration

All options that can be supplied to the application via the command line, can also be specified in the configuration file. Configuration files are simply python files that define the options as variables. The below is an example configuration.

```
# General settings
port = 8888

# One worker process per CPU core
workers = 0

# Set client name and key if the application requires signed requests. The
# client must sign the request using the client_key, see README for
# instructions.
client_name = "sample"
client_key = "3NdajqH8mBLokepU4I2Bh6KK84GUf1lzjnuTdskY"

# Set the allowed hosts as an alternative to signed requests. Only those
# images which are served from the following hosts will be requested.
allowed_hosts = ["localhost"]

# Request-related settings
max_requests = 50
timeout = 7.5

# Set default resizing options
background = "ccc"
filter = "bilinear"
mode = "crop"
position = "top"

# Set default rotating options
expand = False

# Set default saving options
format = None
```

(continues on next page)

(continued from previous page)

```
optimize = 1
quality = "90"
```

CHAPTER 7

Tools

To verify that your client application is generating correct signatures, use the signature command.

```
$ python -m pilbox.signature --key=abcdef "x=1&y=2&z=3"
Query String: x=1&y=2&z=3
Signature: c9516346abf62876b6345817dba2f9a0c797ef26
Signed Query String: x=1&y=2&z=3&sig=c9516346abf62876b6345817dba2f9a0c797ef26
```

The application allows the use of the resize functionality via the command line.

```
$ python -m pilbox.image --width=300 --height=300 http://i.imgur.com/zZ8XmBA.jpg > /
↪tmp/foo.jpg
```

If a new mode is added or a modification was made to the libraries that would change the current expected output for tests, run the generate test command to regenerate the expected output for the test cases.

```
$ python -m pilbox.test.genexpected
```

CHAPTER 8

Deploying

It is strongly encouraged that pilbox not be directly accessible to the internet. Instead, it should only be accessible via a web server, e.g. NGINX or Apache, or some other application that is designed to handle direct traffic from the internet.

The application itself does not include any caching. It is recommended that the application run behind a CDN for larger applications or behind varnish for smaller ones.

Defaults for the application have been optimized for quality rather than performance. If you wish to get higher performance out of the application, it is recommended you use a less computationally expensive filtering algorithm and a lower JPEG quality. For example, add the following to the configuration.

```
# Set default resizing options
filter = "bicubic"
quality = 75
```

If you wish to improve performance further and are using an x86 platform, you may want to consider using Pillow-SIMD. Follow the steps in Installation and it should function as a drop-in replacement for `Pillow`. To avoid any incompatibility issues, use the same version of `Pillow-SIMD` as is being used for `Pillow`.

Another setting that's helpful for fine-tuning performance and memory usage is the `workers` setting to set the number of Tornado worker processes. The default setting of `0` spawns one worker process per CPU core which can lead to high memory usage and reduced performance due to swapping on low-memory configurations. For Heroku deployments limiting the number of worker processes to 2-3 for the lower-end dynos helped smooth out application response time.

CHAPTER 9

Extension

While it is generally recommended to use Pilbox as a standalone server, it can also be used as a library. To extend from it and build a custom image processing server, use the following example.

```
#!/usr/bin/env python

import tornado.gen

from pilbox.app import PilboxApplication, ImageHandler, \
    start_server, parse_command_line


class CustomApplication(PilboxApplication):
    def get_handlers(self):
        return [(r"/(\d+)x(\d+)/(.+)", CustomImageHandler)]


class CustomImageHandler(ImageHandler):
    def prepare(self):
        self.args = self.request.arguments.copy()

    @tornado.gen.coroutine
    def get(self, w, h, url):
        self.args.update(dict(w=w, h=h, url=url))

        self.validate_request()
        resp = yield self.fetch_image()
        self.render_image(resp)

    def get_argument(self, name, default=None):
        return self.args.get(name, default)


if __name__ == "__main__":
    parse_command_line()
    start_server(CustomApplication())
```

CHAPTER 10

Contribution

To contribute to the project or to make and test your own changes, fork and then clone the project.

```
$ git clone https://github.com/YOUR-USERNAME/pilbox.git
```

Packaged with Pilbox is a Vagrant configuration file which installs all necessary dependencies on a virtual box using Ansible. See the Vagrant documentation and the Ansible documentation for installation instructions. Once installed, the following will start and provision a virtual machine.

```
$ vagrant up
$ vagrant provision
```

This will have installed pilbox in `/var/www/pilbox` on the virtual machine. To access the virtual machine itself, simply...

```
$ vagrant ssh
```

When running via Vagrant, the application is automatically started on port 8888 on 192.168.100.100, i.e.

```
http://192.168.100.100:8888/
```

To run pilbox manually, execute the following.

```
$ sudo /etc/init.d/pilbox stop
$ python -m pilbox.app
```

To run all tests, issue the following command from the installed pilbox directory.

```
$ python -m pilbox.test.runtests
```

To run individual tests, simply indicate the test to be run, e.g.

```
$ python -m pilbox.test.runtests pilbox.test.signature_test
```

CHAPTER 11

Changelog

- 0.1: Image resizing fit
- 0.1.1: Image cropping
- 0.1.2: Image scaling
- 0.2: Configuration integration
- 0.3: Signature generation
- 0.3.1: Signature command-line tool
- 0.4: Image resize command-line tool
- 0.5: Facial recognition cropping
- 0.6: Fill resizing mode
- 0.7: Resize using crop position
- 0.7.1: Resize using a single dimension, maintaining aspect ratio
- 0.7.2: Added filter and quality options
- 0.7.3: Support python 3
- 0.7.4: Fixed cli for image generation
- 0.7.5: Write output in 16K blocks
- 0.8: Added support for ARGB (alpha-channel)
- 0.8.1: Increased max clients and write block sizes
- 0.8.2: Added configuration for max clients and timeout
- 0.8.3: Only allow http and https protocols
- 0.8.4: Added support for WebP
- 0.8.5: Added format option and configuration overrides for mode and format
- 0.8.6: Added custom position support

- 0.9: Added rotate operation
- 0.9.1: Added sub-region selection operation
- 0.9.4: Added Pilbox as a PyPI package
- 0.9.10: Converted README to reStructuredText
- 0.9.14: Added Sphinx docs
- 0.9.15: Added implicit base url to configuration
- 0.9.16: Added validate cert to configuration
- 0.9.17: Added support for GIF format
- 0.9.18: Fix for travis builds on python 2.6 and 3.3
- 0.9.19: Validate cert fix
- 0.9.20: Added optimize option
- 0.9.21: Added console script entry point
- 1.0.0: Modified for easier library usage
- 1.0.1: Added allowed operations and default operation
- 1.0.2: Modified to allow override of http content type
- 1.0.3: Safely catch image save errors
- 1.0.4: Added progressive option
- 1.1.0: Proxy server support
- 1.1.1: Added JPEG auto rotation based on Exif orientation
- 1.1.2: Added keep JPEG quality option and set JPEG subsampling to keep
- 1.1.3: Fixed auto rotation on JPEG with missing Exif data
- 1.1.4: Exception handling around invalid Exif data
- 1.1.5: Fixed image requests without content types
- 1.1.6: Support custom applications that need command line arguments
- 1.1.7: Support adapt resize mode
- 1.1.8: Added preserve Exif flag
- 1.1.9: Increased Pillow version to 2.8.1
- 1.1.10: Added ca_certs option
- 1.1.11: Added support for TIFF
- 1.2.0: Support setting background when saving a transparent image
  - *Backwards incompatible*: default background property changed to `0fff`. To restore previous behavior, set background in config to `ffff`.
- 1.2.1: Added max operations config property
- 1.2.2: Added max resize width and height config properties
- 1.2.3: Added user_agent option
- 1.3.0: Increased Pillow to 2.9.0 and Tornado to 4.5.1

- 1.3.1: Fix pilbox.image CLI for python 3.0
- 1.3.2: Fix GIF P-mode to JPEG conversion
- 1.3.3: Increase Pillow version to 5.2.0 and Tornado version to 5.1.0
- 1.3.4: Added worker config property to set number of Tornado processes