Thumbor Documentation

Release 7.8.0

Bernardo Heynemann

Nov 14, 2026

CONTENTS

1 What's Thumbor? 3

2 Contents 5

2.1 Installing . . . 5
2.2 Getting Started . . . 6
2.3 Usage . . . 7
2.4 Imaging . . . 11
2.5 Customizing Thumbor . . . 57
2.6 Administration . . . 75
2.7 Upload . . . 100
2.8 Contributors & Users . . . 106

3 Indices and tables 111

Python Module Index 113

Index 115


thumbor

CHAPTER

ONE

WHAT'S THUMBOR?

Thumbor is a smart imaging service. It enables on-demand crop, resizing and flipping of images.

It features a VERY smart detection of important points in the image for better cropping and resizing, using state-of-the-art face and feature detection algorithms (more on that in *Detection Algorithms*).

Using thumbor is very easy (after it is running). All you have to do is access it using an URL for an image, like this:

```
http://thumbor-server/unsafe/300x200/smart/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor
↪%2Fraw%2Fmaster%2Fexample.jpg
```

That URL would show an image of the Big Brother Brasil participants in 300x200 using smart crop. There are several other options to the image URL configuration. You can check them in the *Usage* page. For more details on the /unsafe part of the URL, check the *Security* page.

The safe url for the above URL would look like (check *Security* for more details):

```
http://thumbor-server/K97LekICOXT9MbO3X1u8BBkrjbu5/300x200/smart/https%3A%2F%2Fgithub.com
↪%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```

Warning

Release 7.0.0 introduces a major breaking change due to the migration to python 3 and the modernization of our codebase. Please read the release notes for details on how to upgrade.

CHAPTER

TWO

CONTENTS

2.1 Installing

thumbor is distributed as a Python package and can be installed with any standards-compliant Python package installer.

> **Warning**
>
> Current thumbor releases require Python 3.11 or newer. Please refer to release notes for details on how to upgrade.

2.1.1 Stable

The latest stable version of thumbor is published on the Python Package Index. Install it with:

```
python -m pip install thumbor
```

2.1.2 From the source of a stable release

Download the latest stable source-code version here on GitHub or PyPI and decompress it.

From the directory containing the extracted source, run:

```
python -m pip install .
```

2.1.3 From the latest version of the source

Install the latest source directly from GitHub:

```
python -m pip install "git+https://github.com/thumbor/thumbor.git"
```

Alternatively, clone the repository and install it locally:

```
git clone https://github.com/thumbor/thumbor.git
cd thumbor
python -m pip install .
```

Contributors should use `make setup` instead; it creates the locked uv development environment with the project's test and optional dependencies.

2.2 Getting Started

If you just want to give thumbor a try, it is pretty easy to get started. **It won't take more than a minute.**

Just install it with `pip install thumbor` and start the process with `thumbor` in a console. That's all you need to start transforming images.

The image we'll be using in most of our examples is a Creative Commons licensed image by Snapwire:

```
https://github.com/thumbor/thumbor/raw/master/example.jpg
```



If you want to use a different image, go ahead. Any image will work for the remainder of the docs.

Note

Thumbor only understands properly encoded URIs. In order to use the URI above (or any other for that matter), we first need to encode it. This can be easily achieved by going to any modern browser's developer console and typing:

```
window.encodeURIComponent(
  "https://github.com/thumbor/thumbor/raw/master/example.jpg"
)
```

And the output will be:

```
https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```

That's the URL we'll be using in our examples!

2.2.1 Problems installing thumbor locally

No worries! If you have a docker host accessible, just run:

```
$ docker run -p 8888:80 ghcr.io/minimalcompact/thumbor:latest
```

After downloading the image and running it, thumbor will be accessible at `http://localhost:8888/`.

For more details, see the MinimalCompact image.

2.2.2 Changing its size

Go to your browser and enter in the url:

```
http://localhost:8888/unsafe/300x200/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw
↪%2Fmaster%2Fexample.jpg
```

You should see the image with $300px$ of width and $200px$ of height. Just play with it in the url to see the image change.

If you just want it to be proportional to the width, enter a height of 0, like:

```
http://localhost:8888/unsafe/300x0/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw
↪%2Fmaster%2Fexample.jpg
```

2.2.3 Flipping the image

How about seeing it backwards? Or upside down?

Go to your browser and enter in the url:

```
http://localhost:8888/unsafe/-0x-0/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw
↪%2Fmaster%2Fexample.jpg
```

You should see the waterfall backwards and upside down.

2.2.4 Filters

What if I want to change contrast or brightness?

Go to your browser and enter in the url:

```
http://localhost:8888/unsafe/filters:brightness(10):contrast(30)/https%3A%2F%2Fgithub.com
↪%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```

There are many more filters to explore. Check the *Filters* page for more details.

2.2.5 What now?

Ok, now that you know how amazing thumbor is, there's actually A LOT more to it. Go check the rest of the docs to learn how to get even more from your new imaging server.

2.3 Usage

Using thumbor is really straightforward. thumbor offers one endpoint for retrieving the image and a very similar endpoint to retrieve metadata.

2.3.1 Image Endpoint

```
/hmac/trim/AxB:CxD/(adaptive-)(full-)fit-in/-Ex-F/HALIGN/VALIGN/smart/
↪filters:FILTERNAME(ARGUMENT):FILTERNAME(ARGUMENT)/*IMAGE-URI*
```

- hmac is the signature that ensures *Security* ;
- trim removes surrounding space in images using top-left pixel color unless specified otherwise;
- AxB:CxD means manually crop the image at left-top point AxB and right-bottom point CxD;
- fit-in means that the generated image should not be auto-cropped and otherwise just fit in an imaginary box specified by ExF. If a full fit-in is specified, then the largest size is used for cropping (width instead of height, or the other way around). If adaptive fit-in is specified, it inverts requested width and height if it would get a better image definition;
- -Ex-F means resize the image to be ExF of width per height size. The minus signs mean flip horizontally and vertically;

- HALIGN is horizontal alignment of crop;
- VALIGN is vertical alignment of crop;
- smart means using smart detection of focal points;
- filters can be applied sequentially to the image before returning;
- *IMAGE-URI* is the encoded URI for the image you want resized.

Trim

Removing surrounding space in images can be done using the trim option.

Unless specified trim assumes the top-left pixel color and no tolerance (more on tolerance below).

To use it, just add a `/trim` part to your URL.

If you need to specify the orientation from where to get the pixel color, just use `/trim:top-left` for the top-left pixel color or `/trim:bottom-right` for the bottom-right pixel color.

Trim also supports color tolerance. The euclidean distance between the colors of the reference pixel and the surrounding pixels is used. If the distance is within the tolerance they'll get trimmed. For a RGB image the tolerance would be within the range 0-442.

Manual Crop

The manual crop is entirely optional. This is very useful for applications that provide custom real-time cropping capabilities to their users.

The manual crop part of the url takes two points as arguments, separated by a colon. The first point is the left-top point of the cropping rectangle. The second point is the right-bottom point.

This crop is performed before the rest of the operations, so it can be used as a *prepare* step before resizing and smart-cropping. It is **very** useful when you just need to get that celebrity face on a big picture full of people, as an example.

Fit in

The fit-in argument specifies that the image should not be auto-cropped and auto-resized to be **EXACTLY** the specified size, and should be fit in an imaginary box of "E" width and "F" height, instead.

Consider an image of $800px$ x $600px$, and a fit of $300px$ x $200px$. This is how thumbor would resize it:



Consider an image of $400px$ x $600px$, and a fit of $300px$ x $200px$. This is how thumbor would resize it:



This is very useful when you need to fit an image somewhere, but you have no idea about the original image dimensions.

If a full fit-in is used, instead of using the largest size for cropping it uses the smallest one, so in the above scenarios:

For the image of $800px$ x $600px$, with a full fit-in of $300px$ x $200px$, we would get an image of $300px$ x $225px$.

For the image of $400px$ x $600px$, with a full fit-in of $300px$ x $200px$, we would get an image of $300px$ x $450px$.

Image Size

The image size argument specifies the size of the image that will be returned by the service. Thumbor uses smart *Crop and Resize Algorithms*

If you omit one of the dimensions or use zero as a value (as in $300x$, $300x0$, $x200$, $0x200$, and so on), Thumbor will determine that dimension as to be proportional to the original image. Say you have an $800x600$ image and ask for a $400x0$ image. Thumbor will infer that since 400 is half of 800, then the height you are looking for is half of 600, which is $300px$.

If you use $0x0$, Thumbor will use the original size of the image and thus won't do any cropping or resizing.

If you specify one of the dimensions as the string "orig" (as in $origx100$, $100xorig$, $origxorig$), thumbor will interpret that you want that dimension to remain the same as in the original image. Consider an image of $800x600$. If you ask for a $300xorig$ version of it, thumbor will interpret that you want a $300x600$ image. If you instead ask for a $origx300$ version, thumbor will serve you an $800x300$ image.

If you use $origxorig$, Thumbor will use the original size of the image and thus won't do any cropping or resizing.

The default value (in case it is omitted) for this option is to use proportional size (0) to the original image.

Horizontal Align

As was explained above, unless the image is of the same proportion as the desired size, some cropping will need to occur.

The horizontal align option controls where the cropping will occur if some width needs to be trimmed (unless some feature detection occurs - more on that later).

So, if we need to trim $300px$ of the width and the current horizontal align is "left", then we'll trim 0px of the left of the image and $300px$ of the right side of the image.

The possible values for this option are:

- `left` - only trims the right side;
- `center` - trims half of the width from the left side and half from the right side;
- `right` - only trims the left side.

It is important to notice that this option is useless in case of the image being vertically trimmed, since Thumbor's cropping algorithm only crops in one direction.

The default value (in case it is omitted) for this option is "center".

Vertical Align

The vertical align option is analogous to the horizontal one, except that it controls height trimming.

So, if we need to trim $300px$ of the height and the current vertical align is "top", then we'll trim $0px$ of the top of the image and $300px$ of the bottom side of the image.

The possible values for this option are:

- `top` - only trims the bottom;
- `middle` - trims half of the height from the top and half from the bottom;
- `bottom` - only trims the top.

It is important to notice that this option is useless in case of the image being horizontally trimmed, since Thumbor's cropping algorithm only crops in one direction.

The default value (in case it is omitted) for this option is "middle".

Smart Cropping

Thumbor uses some very advanced techniques for obtaining important points of the image (referred to as Focal Points in the rest of this documentation).

Even though Thumbor comes with facial recognition of Focal Points as well as feature recognition, you can easily implement your own detectors as you'll see further in the docs.

There's not much to this option, since we'll cover it in the *Detection Algorithms* page. If you use it in the url, smart cropping will be performed and will override both horizontal and vertical alignments if it finds any Focal Points.

The default value (in case it is omitted) for this option is not to use smart cropping.

Filters

Thumbor allows for usage of a filter pipeline that will be applied sequentially to the image. Filters are covered in the *Filters* page if you want to know more.

To use filters add a `filters:` part in your URL. Filters are like function calls `filter_name(argument, argument2, etc)` and are separated using the `:` character, like `filters:filter_name():other_filter()`.

Image URI

This is the image URI. The format of this option depends heavily on the image loader you are using. Thumbor comes pre-packaged with an HTTP loader and a Filesystem loader.

If you use the HTTP loader, this option corresponds to the image complete URI.

If you use the Filesystem loader, this option corresponds to the path of the image from the images root.

You can learn more about the loaders in the *Image loader* page.

2.3.2 Metadata Endpoint

The metadata endpoint has **ALL** the options that the image one has, but instead of actually performing the operations in the image, it just simulates the operations.

Since it has the same options as the other endpoint, we won't repeat all of them. To use the metadata endpoint, just add `/meta` in the beginning of the url.

Say we have the following crop URL:

`http://my-server.thumbor.org/unsafe/-300x-200/left/top/smart/path/to/my/nice/image.jpg`

If we want the metadata on what thumbor would do, just change the url to be

`http://my-server.thumbor.org/unsafe/meta/-300x-200/left/top/smart/path/to/my/nice/image.jpg`

After the processing is finished, thumbor will return a json object containing metadata on the image and the operations that would have been performed.

The json looks like this:

```
{
    thumbor: {
        source: {
            url: "path/to/my/nice/image.jpg",
            width: 800,
            height: 600
        },
        operations: [
            {
                type: "crop",
                left: 10,
                top: 10,
                right: 300,
                bottom: 200
            },
            {
                type: "resize",
                width: 300,
                height: 200
            },
            { type: "flip_horizontally" },
            { type: "flip_vertically" }
        ]
    }
}
```

2.4 Imaging

2.4.1 Crop and Resize Algorithms

Note

thumbor performs the least amount of cropping possible to resize your image to the exact size you specified, without changing it's aspect ratio.

Cropping the image

Before resizing the image, thumbor crops it so it has the same aspect as the desired dimensions. Let's see an example to clarify this concept.

Consider an $800x600$ (width x height, in pixels) image and say we want a $400x150$ thumbnail of it. The first thing thumbor needs to do is calculate the proportion of the images:

$$width : 800600 = 1.333$$

$$height : 400150 = 2.666$$

Now that they don't match, thumbor defines if the image needs horizontal or vertical cropping. We never crop both ways, since it's not needed.

So, in our example to get an image of the same proportion of the target one, we need to get the picture height to be $300px$ (using the proportional height):

$$h = 800x150400 = 300$$

Now all we need to do is cropping $300px$ of the picture height. To determine whether to crop from the top, bottom or both we use the focal points or the horizontal alignment. If any focal points have been specified we'll use those to find the center of mass of the image (more on that in *Detection Algorithms*). Otherwise we'll use the horizontal and vertical alignments.



Let's say that for this image no focal points were found, so we'll use the vertical alignment to crop the height. Since we specified middle alignment for this example, we'll crop off $150px$ from the top and $150px$ from the bottom of the image, similarly to this image:



```
http://localhost:8888/unsafe/300x300/filters:vertical_align(middle)/https%3A%2F%2Fgithub.
↪com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```

Here's an example of how thumbor would crop width or height using centered alignment:



```
http://localhost:8888/unsafe/400x600/filters:horizontal_align(center)/https%3A%2F
↪%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



```
http://localhost:8888/unsafe/400x150/filters:vertical_align(middle)/https%3A%2F%2Fgithub.
↪com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```

Resizing the Image

Now that the image has the same proportion as the image we want, it's just a matter of resizing it to the desired dimensions.

Flipping the Image

If the desired dimensions feature negative numbers, thumbor will flip them around that direction. This means that negative width specifies horizontal flip, while negative height specifies vertical flip.

2.4.2 Filters

How Filters Work

Thumbor handles filters in a pipeline. This means that they run sequentially in the order they are specified! Given an original image with size $60x40$ and the following transformations:

```
http://localhost:8888/fit-in/100x100/filters:watermark(..):blur(..):fill(red,
↪1):upscale()/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```

The resulting image will first check if it can fit into a $100x100$. Since it does, the filter pipeline will kick in and:

- add the watermark in the image;
- blur the whole image (including the watermark);
- Fill the outer parts of the image with red (so it will fit in $100x100$);
- Then it will try to upscale. This will have no effect, since at this point the image is already $100x100$.

Available Filters

AutoJPG

Usage: `autojpg(enabled)`

Description

This filter overrides only the `AUTO_PNG_TO_JPG` fallback for the current request. It does not enable or disable a `jpg` entry explicitly configured in `AUTO_IMAGE_FORMAT_PREFERENCE`.

Arguments

- `enabled` - `True` enables PNG-to-JPEG fallback and `False` disables it for this request. The default is `True`.

Example

```
http://localhost:8888/unsafe/300x300/filters:autojpg()/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```

Background Color

Usage: `background_color(color)`

Description

The background_color filter sets the background layer to the specified color. This is specifically useful when converting transparent images (PNG) to JPEG

Arguments

- `color` - the color name (like in HTML) or hexadecimal rgb expression without the "#" character. See the list of supported web colors. If color is "auto", a color will be smartly chosen based on the image pixels.

Example

The original image is:



```
http://localhost:8888/unsafe/fit-in/300x300/filters:background_color(blue)/https%3A%2F
↪%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fdocs%2Fimages%2Fdice_transparent_
↪background.png
```



```
http://localhost:8888/unsafe/fit-in/300x300/filters:background_color(f00)/https%3A%2F
↪%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fdocs%2Fimages%2Fdice_transparent_
↪background.png
```



```
http://localhost:8888/unsafe/fit-in/300x300/filters:background_color(add8e6)/https%3A%2F
↪%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fdocs%2Fimages%2Fdice_transparent_
↪background.png
```



Blur

Usage: `blur(radius [, sigma])`

Description

This filter applies a gaussian blur to the image.

Arguments

- `radius` - Radius used in the gaussian function to generate a matrix, maximum value is 150. The bigger the radius more blurred will be the image.
- `sigma` - Optional. Defaults to the same value as the radius. Sigma used in the gaussian function.

Example



```
http://localhost:8888/unsafe/filters:blur(7)/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor
↪%2Fraw%2Fmaster%2Fexample.jpg
```



Brightness

Usage: `brightness(amount)`

Description

This filter increases or decreases the image brightness.

Arguments

- `amount` - `-100 to 100` - The amount (in %) to change the image brightness. Positive numbers make the image brighter and negative numbers make the image darker.

Example



```
http://localhost:8888/unsafe/filters:brightness(40)/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Contrast

Usage: `contrast(amount)`

Description

This filter increases or decreases the image contrast.

Arguments

- `amount` - -100 to 100 - The amount (in %) to change the image contrast. Positive numbers increase contrast and negative numbers decrease contrast.

Example



```
http://localhost:8888/unsafe/filters:contrast(40)/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



```
http://localhost:8888/unsafe/filters:contrast(-40)/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Convolution

Usage: `convolution(matrix_items, number_of_columns, should_normalize)`

Description

This filter runs a convolution matrix (or kernel) on the image. See Kernel (image processing) for details on the process. Edge pixels are always extended outside the image area.

Arguments

- `matrix_items` - Semicolon separated integer or decimal matrix items. Decimal values must include digits after the decimal point, e.g. `1.5`.
- `number_of_columns` - Number of columns in the matrix.
- `should_normalize` - Whether or not we should divide each matrix item by the sum of all items.

Example



Normalized Matrix:

```
1 2 1
2 4 2
2 1 2
```

```
http://localhost:8888/unsafe/filters:convolution(1;2;1;2;4;2;1;2;1,3,true)/http://upload.
↪wikimedia.org/wikipedia/commons/5/50/Vd-Orig.png
```



Matrix:

```
-1 -1 -1
-1  8 -1
-1 -1 -1
```

```
http://localhost:8888/unsafe/filters:convolution(-1;-1;-1;-1;8;-1;-1;-1;-1,3,false)/
↪http://upload.wikimedia.org/wikipedia/commons/5/50/Vd-Orig.png
```



Cover

Usage: `cover()`

Description

This filter is used in GIFs to extract their first frame as the image to be used as cover.

> **Note**
>
> This filter will only function when `USE_GIFSICLE_ENGINE` is set to `True` in `thumbor.conf`:

```
USE_GIFSICLE_ENGINE = True
```

Arguments

No arguments.

Example

```
http://localhost:8888/unsafe/filters:cover()/http://server.my/animated_static.gif
```

Equalize

Usage: `equalize()`

Description

This filter equalizes the color distribution in the image.

Arguments

No arguments.

Example



```
http://localhost:8888/unsafe/filters:equalize()/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Extract focal points

Usage: `extract_focal()`

Description

When cropping, thumbor uses focal points in the image to direct the area of the image that matters most. There are several ways of finding focal points. To learn more about focal points, visit the *Detection Algorithms*.

In order to use the `extract_focal` filter, the original image must be a thumbor URL that features manual cropping. To learn more about manual cropping, visit the *Crop and Resize Algorithms*.

Using the original manual cropping points, this filter adds the cropped area (originally in the format `/LEFTxTOP:RIGHTxBOTTOM/`) as a focal point for the new image.

For the new image, thumbor will use as the original the image URL that was the original for the segment with the manual cropping.

This means that for an URL like:

```
http://localhost:8888/unsafe/300x100/filters:extract_focal()/localhost:8888/unsafe/
↪240x220:480x260/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.
↪jpg
```

Thumbor will use as original the following image URL:

```
https://github.com/thumbor/thumbor/raw/master/example.jpg
```

Example

Original Image:



Eye cropped:

```
http://localhost:8888/unsafe/240x220:480x260/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor
↪%2Fraw%2Fmaster%2Fexample.jpg
```



A bigger image based on above's crop with the extract_focal() filter:

```
http://localhost:8888/unsafe/300x100/filters:extract_focal()/localhost:8888/unsafe/
↪240x220:480x260/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.
↪jpg
```



Without the filter that would be the result:

```
http://localhost:8888/unsafe/300x100/localhost:8888/unsafe/240x220:480x260/https%3A%2F
↪%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Filling

Usage: `fill(color[,fill_transparent])`

Description

This filter returns an image sized exactly as requested independently of its ratio. It will fill the missing area with the specified color. It is usually combined with the "fit-in" or "adaptive-fit-in" options.

Arguments

- `color` - the color name (like in HTML) or hexadecimal RGB expression without the "#" character. See the list of supported web colors.

  If color is "transparent" and the image format, supports transparency the filling color is transparent.

  > **Warning**
  >
  > Some engines (like OpenCV engine) do not support transparency.

  If color is "auto", a color is smartly chosen (based on the image pixels) as the filling color.

  If color is "blur", the missing parts are filled with blurred original image.

- `fill_transparent` - a boolean to specify whether transparent areas of the image should be filled or not. Accepted values are either `true`, `false`, `1` or `0`. This argument is optional and the default value is `false`.

Example #1

The original image is:



```
http://localhost:8888/unsafe/fit-in/300x300/filters:fill(blue)/https%3A%2F%2Fgithub.com
↪%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



```
http://localhost:8888/unsafe/fit-in/300x300/filters:fill(f00)/https%3A%2F%2Fgithub.com
↪%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



```
http://localhost:8888/unsafe/fit-in/300x300/filters:fill(add8e6)/https%3A%2F%2Fgithub.com
↪%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



```
http://localhost:8888/unsafe/fit-in/300x300/filters:fill(auto)/https%3A%2F%2Fgithub.com
↪%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



```
http://localhost:8888/unsafe/fit-in/300x300/filters:fill(blur)/https%3A%2F%2Fgithub.com
↪%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Example #2

The original image is:



```
http://localhost:8888/unsafe/fit-in/300x225/filters:fill(blue,1)/https://github.com/
↪thumbor/thumbor/wiki/dice_transparent_background.png
```



```
http://localhost:8888/unsafe/fit-in/300x225/filters:fill(f00,true)/https://github.com/
↪thumbor/thumbor/wiki/dice_transparent_background.png
```



```
http://localhost:8888/unsafe/fit-in/300x225/filters:fill(add8e6,1)/https://github.com/
↪thumbor/thumbor/wiki/dice_transparent_background.png
```



```
http://localhost:8888/unsafe/fit-in/300x225/filters:fill(auto,true)/https://github.com/
↪thumbor/thumbor/wiki/dice_transparent_background.png
```



```
http://localhost:8888/unsafe/fit-in/300x225/filters:fill(blur,true)/https://github.com/
↪thumbor/thumbor/wiki/dice_transparent_background.png
```



Focal

Usage: `focal(<left>x<top>:<right>x<bottom>)`

Description

This filter adds a focal point, which is used in later transforms.

Arguments

- `left, top, right, bottom`: All mandatory arguments in the `<left>x<top>:<right>x<bottom>` format.

Example

Before cropping with specific focal point:



```
http://localhost:8888/unsafe/400x100/filters:focal(146x206:279x360)/https%3A%2F%2Fgithub.
↪com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```

After specifying the focal point:



> **Warning**
>
> When using this filter together with detectors, the extract focal points filter, or the metadata parameter, unexpected behavior may occur.

Format

Usage: `format(image-format)`

Description

This filter specifies the output format of the image. The output must be one of: "webp", "jpeg", "gif", "png", "avif" or "heic".

Arguments

- `image-format` - The output format of the resulting image.

Example

```
http://localhost:8888/unsafe/filters:format(webp)/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```

Grayscale

Usage: `grayscale()`

Description

This filter changes the image to grayscale.

Arguments

No arguments.

Example



```
http://localhost:8888/unsafe/filters:grayscale()/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Max bytes

Usage: `max_bytes(number-of-bytes)`

Description

This filter automatically degrades the quality of the image until the image is under the specified amount of bytes.

Arguments

- `number-of-bytes` - The maximum number of bytes for the given image.

Example

Compressing the original image to less than 7.5k (ended up with ~7kb):



```
http://localhost:8888/unsafe/filters:max_bytes(7500)/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



No upscale

Usage: `no_upscale()`

Description

This filter tells thumbor not to upscale your images.

This means that if an original image is $300px$ width by $200px$ height and you ask for a $600x400$ image, thumbor will still return a $300x200$ image.

When the URL also crops or trims the image, the limit is the area left after cropping and trimming, so that area is never upscaled either.

Arguments

No arguments allowed.

Example

```
http://localhost:8888/unsafe/filters:no_upscale()/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```

Noise

Usage: `noise(amount)`

Description

This filter adds noise to the image.

Arguments

- `amount` - `0% to 100%` - The amount of noise to add to the image.

Example



```
http://localhost:8888/unsafe/filters:noise(40)/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Proportion

Usage: `proportion(percentage)`

Description

This filter applies the specified proportion to the image's height and width when cropping.

Arguments

- `percentage` - The float percentage of the proportion (0.0 to 1.0).

Example



```
http://localhost:8888/unsafe/filters:proportion(0.5)/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Quality

Usage: `quality(amount)`

Description

This filter changes the overall quality of the JPEG image (does nothing for PNGs or GIFs).

Arguments

- `amount` - `0 to 100` - The quality level (in %) that the end image will feature.

Example



```
http://localhost:8888/unsafe/filters:quality(40)/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Red eye

Not documented yet

RGB

Usage: `rgb(rAmount, gAmount, bAmount)`

Description

This filter changes the amount of color in each of the three channels.

Arguments

- `rAmount` - The amount of redness in the picture. Can range from -100 to 100 in percentage.
- `gAmount` - The amount of greenness in the picture. Can range from -100 to 100 in percentage.
- `bAmount` - The amount of blueness in the picture. Can range from -100 to 100 in percentage.

Example



```
http://localhost:8888/unsafe/filters:rgb(20,-20,40)/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Rotate

Usage: `rotate(angle)`

Description

This filter rotates the given image according to the angle value passed.

> **Note**
>
> This filter rotates the image according to the engine. For the PIL engine the rotation is done counter-clockwise.

Arguments

- `angle` - `0 to 359` - The euler angle to rotate the image by. Numbers greater or equal than 360 will be transformed to an equivalent angle between 0 and 359.

Example



```
http://localhost:8888/unsafe/filters:rotate(90)/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Round corners

Usage: `round_corner(a|b,r,g,b,[transparent])`

Description

This filter adds rounded corners to the image using the specified color as background.

Arguments

- `a|b` - amount of pixels to use as radius. The argument `b` is not required, but it specifies the second value for the ellipsis used for the radius.
- `transparent` - Optional. If set to true/1, the background will be transparent and the rounded edge is smoothed through the alpha channel. Use an output format with alpha, such as PNG, WebP or AVIF. Formats without alpha, such as JPEG, show the color in the corners with a hard edge, so omit `transparent` there to blend the edge into the color.

Examples



```
http://localhost:8888/unsafe/filters:round_corner(20,255,255,255)/https%3A%2F%2Fgithub.
↪com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



```
http://localhost:8888/unsafe/filters:round_corner(20|40,0,0,0)/https%3A%2F%2Fgithub.com
↪%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



```
http://localhost:8888/unsafe/filters:round_corner(30,0,0,0,1):format(png)/https%3A%2F
↪%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Saturation

Usage: `saturation(amount)`

Description

This filter increases or decreases the image saturation.

Arguments

- `amount` - -100 to 100 - The amount (in %) to change the image saturation. Positive numbers increase saturation and negative numbers decrease saturation.

Example



http://localhost:8888/unsafe/filters:saturation(40)/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg



http://localhost:8888/unsafe/filters:saturation(-40)/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg



Sharpen

Usage: `sharpen(sharpen_amount,sharpen_radius,luminance_only)`

Description

This filter enhances apparent sharpness of the image. It's heavily based on Marco Rossini's excellent Wavelet sharpen GIMP plugin. Check http://registry.gimp.org/node/9836 for details about how it work.

Arguments

- `sharpen_amount` - Sharpen amount. Typical values are between 0.0 and 10.0.
- `sharpen_radius` - Sharpen radius. Typical values are between 0.0 and 2.0.
- `luminance_only` - Sharpen only luminance channel. Values can be `true` or `false`.

Example 1



```
http://localhost:8888/unsafe/filters:sharpen(2,1.0,true)/https%3A%2F%2Fgithub.com
↪%2Fthumbor%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Stretch

Usage: `stretch()`

Description

This filter stretches the image until it fits the required width and height, instead of cropping the image.

Example



```
http://localhost:8888/unsafe/200x100/filters:stretch()/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```



Strip EXIF

Usage: `strip_exif()`

Description

This filter removes any Exif information in the resulting image. To keep the copyright information you have to set the configuration `PRESERVE_EXIF_COPYRIGHT_INFO = True`.

This is useful if you have set the configuration `PRESERVE_EXIF_INFO = True` but still wish to overwrite this behavior in some cases (e.g. for image icons)

Arguments

No arguments

Example

```
http://localhost:8888/unsafe/filters:strip_exif()/http://www.arte.tv/static-epgapi/
↪057460-011-A.jpg
```

Strip ICC

Usage: `strip_icc()`

Description

This filter removes any ICC information in the resulting image. Even though the image might be smaller, removing ICC information may result in loss of quality.

Arguments

No arguments

Example

```
http://localhost:8888/unsafe/filters:strip_icc()/http://videoprocessing.ucsd.edu/~
↪stanleychan/research/pix/Blurred_foreman_0005.png
```

Upscale

Usage: `upscale()`

Description

This filter tells thumbor to upscale your images. This only makes sense with "fit-in" or "adaptive-fit-in".

This means that if an original image is $300px$ width by $200px$ height and you ask for a $600x500$ image, the filter will resize it to $600x400$.

Arguments

No arguments allowed.

Example

```
http://localhost:8888/unsafe/fit-in/600x500/filters:upscale()/https://raw.
↪githubusercontent.com/thumbor/thumbor/e86324e49d7e53acc2a8057e43f3fdd2ca5cea75/docs/
↪images/dice_transparent_background.png
```

Watermark

Usage: `watermark(imageUrl, x, y, alpha [, w_ratio [, h_ratio]])`

Description

This filter adds a watermark to the image. It can be positioned inside the image with the alpha channel specified and optionally resized based on the image size by specifying the ratio (see *Resizing*).

Arguments

- `imageUrl` - Watermark image URL. It is very important to understand that the same image loader that Thumbor uses will be used here. If this URL contains parentheses they MUST be url encoded, since these are the characters Thumbor uses as delimiters for filter parameters.
- `x` - Horizontal position that the watermark will be in. Positive numbers indicate position from the left and negative numbers indicate position from the right. If the value is 'center' (without the single quotes), the watermark will be centered horizontally. If the value is 'repeat' (without the single quotes), the watermark will be repeated

horizontally. If the value is a positive or negative number followed by a 'p' (ex. 20p) it will calculate the value from the image width as percentage

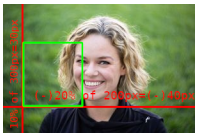
- `y` - Vertical position that the watermark will be in. Positive numbers indicate position from the top and negative numbers indicate position from the bottom. If the value is 'center' (without the single quotes), the watermark will be centered vertically. If the value is 'repeat' (without the single quotes), the watermark will be repeated vertically If the value is a positive or negative number followed by a 'p' (ex. 20p) it will calculate the value from the image height as percentage
- `alpha` - Watermark image transparency. Should be a number between 0 (fully opaque) and 100 (fully transparent).
- `w_ratio` - percentage of the width of the image the watermark should fit-in, defaults to 'none' (without the single quotes) which means it won't be limited in the width on resizing but also won't be resized based on this value. It accepts decimal numbers (ex. 9.5)
- `h_ratio` - percentage of the height of the image the watermark should fit-in, defaults to 'none' (without the single quotes) which means it won't be limited in the height on resizing but also won't be resized based on this value. It accepts decimal numbers (ex. 9.5)

Example

```
http://thumbor-server/filters:watermark(http://my.site.com/img.png,-10,-10,50)/some/
↪image.jpg
```



```
http://thumbor-server/filters:watermark(http://my.site.com/img.png,10p,-20p,50)/some/
↪image.jpg
```



Resizing

Resizing is being done by defining borders the watermark needs to fit in or being upscaled to. The ratio of the watermark will not be changed and will be expanded or shrunk to the size which fits best into the borders. The resulting size is rounded to whole pixels. A side that would round down to zero is kept at 1px, so a watermark shrunk that far only approximates its ratio.

Some examples are shown below with an original image having width=300 and height=200 and an imaginary watermark having width=30 and height=40. Borders are shown in red and the watermark drafted in green.

Considering original image to be 300x200:

- **watermark(imageUrl, 30, 10, 50, 20)**

  20% of the *width*: 300px*0.2 = 60px so the original watermark *width* is 30px which means it can be resized by 2.

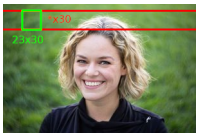
  Because the *height* isn't limited it can grow to 2x40px which is 80px.



- **watermark(imageUrl, 30, 10, 50, none, 15)**

  15% of the *height*: 200px*0.15 = 30px so the original watermark *height* is 40px which means it has to shrink by 25%.

  Because the *width* isn't limited it can shrink to 0.75*30px which is 22.5px (rounded to 23px).



- **watermark(imageUrl, 30, 10, 50, 30, 30)**

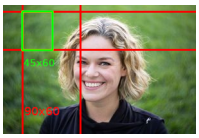
  30% of the *width*: 300px*0.3 = 90px

  and

  30% of the *height*: 200px*0.3 = 60px

  so the original watermark *width* is 30px but cannot use 90px because then (to keep the ratio) the *height* would need to become (40/30)*90px=120px but only 60px is allowed.

  Therefore the *height* is limiting the resizing here and *height* would become 60px and *width* would be (30/40)*60px=45px which fits into the 90px border.



- **watermark(imageUrl, 30, 10, 50, 15.2)**

  15.2% of the *width*: 300px*0.152 = 45.6px, rounded to 46px. Integer ratios would only reach 45px (15%) or 48px (16%).

  Because the *height* isn't limited it grows in the same proportion: (40/30)*45.6px = 60.8px, rounded to 61px.

2.4.3 Detectors

Enabling detectors

Out of the box, thumbor does not enable any feature or facial detection. Enabling it is pretty easy, though.

> **Note**
>
> Starting with release 7.0.0 thumbor depends on opencv-python-headless. This means that it should be extremely easy to use the face and feature detectors.

For information on all built-in detectors check the *Available detectors* page.

Configuration

In order to tell thumbor what detectors it should run in the original image, you must add them to your `thumbor.conf` file in the following key:

```
DETECTORS = [
    'thumbor.detectors.face_detector',
    'thumbor.detectors.feature_detector',
]
```

The above configuration tells thumbor that it should run both the facial detection and the feature detection. These are mutually exclusive, meaning that if a face is detected, the feature detector won't be run.

Using it

After restarting thumbor, it should be as easy as adding a `/smart` option to your URLs, like:

```
http://localhost:8888/unsafe/200x400/smart/https%3A%2F%2Fgithub.com%2Fthumbor%2Fthumbor
↪%2Fraw%2Fmaster%2Fexample.jpg
```

Note

Whenever you are not sure what thumbor is "seeing", use the debug mode:

```
http://localhost:8888/unsafe/debug/200x400/smart/https%3A%2F%2Fgithub.com%2Fthumbor
↪%2Fthumbor%2Fraw%2Fmaster%2Fexample.jpg
```

Thumbor will draw a square on all focal points it found. That way you can be sure of why an image was cropped the way it was.

Lazy Detection

Facial detection can be pretty expensive for thumbor, so it is not advisable to do it synchronously. Please refer to the *Lazy Detection* page for instructions on using it.

Available Detectors

A list of available detectors can be found at *Available detectors*.

Detection Algorithms

If the smart mode of thumbor has been specified in the uri (by the /smart portion of it), thumbor will use it's smart detectors to find focal points.

thumbor comes pre-packaged with two focal-point detection algorithms: facial and feature. First it tries to identify faces and if it can't find any, it tries to identify features (more on that below).

Facial Detection

For instructions on how to get facial detection coordinates see *Metadata Endpoint* .

Note

thumbor uses OpenCV (http://opencv.org) to detect faces. OpenCV returns the rectangle coordinates for the faces it identifies. You can specify the HAAR file Thumbor should use for identification.

Original image



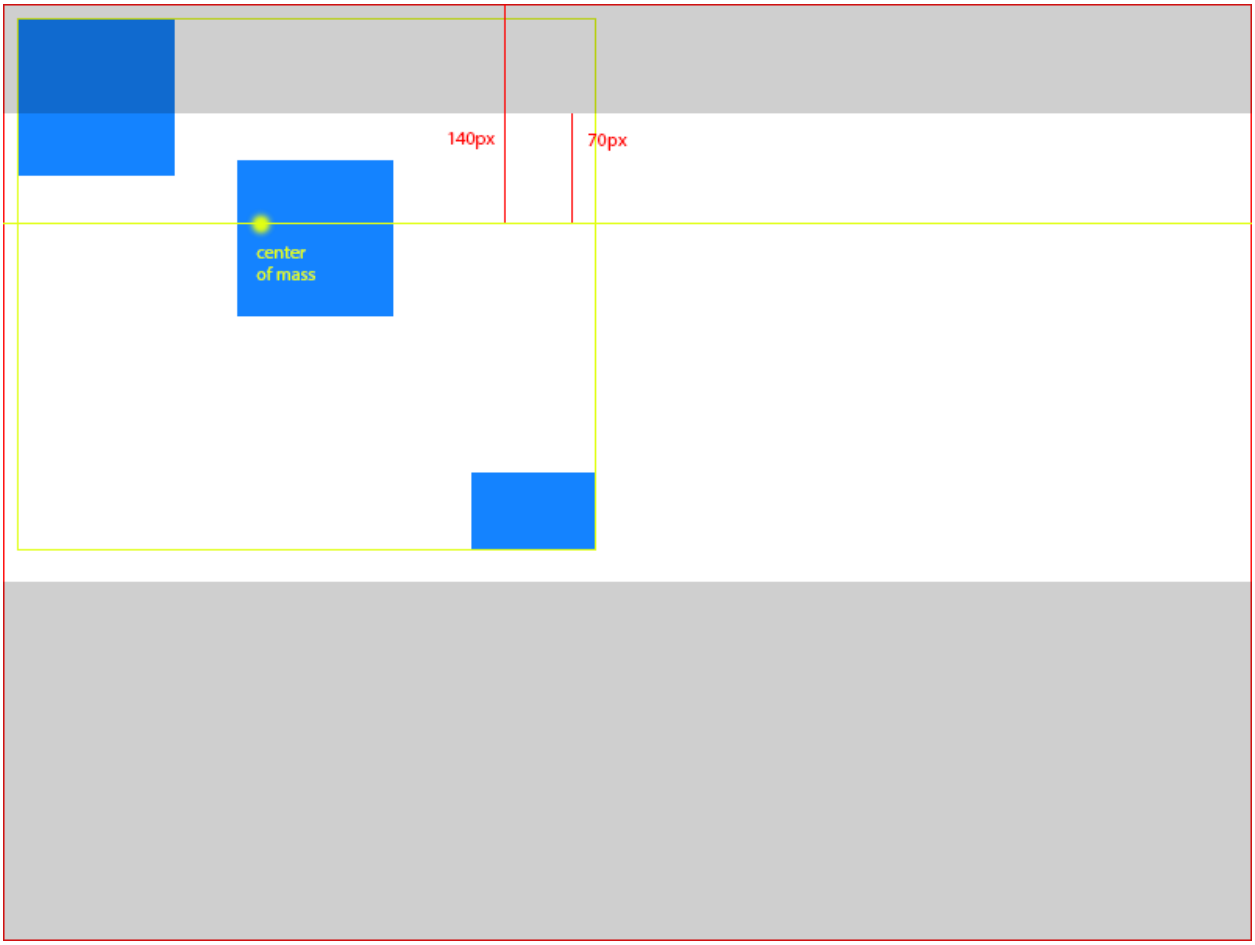
Image after detection

Notice how white rectangle show the area identified as face:



After retrieving these squares from OpenCV, thumbor calculates the *center of mass* of the image using *weighted average*.

Consider that OpenCV returned 3 squares at *(10, 10, 100, 100)*, *(150, 100, 100, 100)*, *(300, 300, 80, 50)*, being *(x, y, width, height)*, as such:



In order to find the *center of mass* for all the faces, we must first find the center and weight of each rectangle. We define weight in this scenario as the area of the rectangle.

So, for the faces in our example (***x***,***y*** being the coordinates of the rectangle's center and ***z*** the rectangle weight):

Face 1:

- $x = (10 + 100)2 = 55$
- $y = (10 + 100)2 = 55$
- $z = 100 * 100 = 10000$

Face 2:

- $x = (150 + 100)2 = 125$
- $y = (100 + 100)2 = 100$
- $z = 100 * 100 = 10000$

Face 3:

- $x = (300 + 80)2 = 190$
- $y = (300 + 50)2 = 175$
- $z = 80 * 50 = 4000$

In order to find the *center of mass* we'll do a *weighted average* of the X and Y coordinates of the faces using:

Horizontal Axis - X: $((55 * 10000) + (125 * 10000) + (190 * 4000))24000 = 106$

Vertical Axis - Y: $((55 * 10000) + (100 * 10000) + (175 * 4000))24000 = 93$

So for the faces found by OpenCV in that image we have the *center of mass* of the picture being $106x93$.

Using Focal Points for Cropping

After finding the center of mass we can use it as the focal point for cropping. Given an image of $800x600$ and a focal point at $106x93$, we need to determine the percentage that needs to be cropped from the top, bottom, left and right sides of the image.

To determine the percentage we use simple math to figure how far from the top and the left side the *center of mass* is:

From the left - $left = 106800 * 100 = 13.25\%$

From the top - $top = 93600 * 100 = 15.50\%$

Using the same example from the *Crop and Resize Algorithms* page, we need to crop $300px$ out of the height of the image. In possession of the percentages of crop above, we can calculate how much we need to crop out of the top and bottom with:

Top - $top = 300 * 0.155 \ = 46$

Bottom - just subtract top $46px$ from the amount of crop (`300px`): $bottom = 300 - 46 = 254$

So, now we now we have to remove $46px$ out of the top of the picture and $254px$ out of the bottom of the picture. In an $800x600$ picture, that means cropping from *(0, 46)* to *(800, 346)*, resulting in an $800x300px$ image.

Assuming we would crop $300px$ horizontally, the cropping would be:

Left - $left = 300 * 0.135 \ = 40$

Right - just subtract left $40px$ from the amount of crop (`300px`): $right = 300 - 40 = 260$

In an image of $800x600$, that means cropping from *(40, 0)* to *(540, 600)*, resulting in a $500x600px$ image. This would not be the case for this image, though.

Feature Detection

If no faces are found in the picture, we still try to find important features in the image, provided by the Good Features to Track Algorithm in OpenCV (http://bit.ly/evAU95).

According to OpenCV documentation, this algorithm finds *"important"* corners in the image. It then returns a list of *(x, y)* values.

We can see the detection taking place in the following images:



The points identified by the good features algorithm:



The cropping based in these features is analogous to the face one, except that all points have a weight of ***1.0*** and are already their centers.

Let's consider that we found **3** feature points: *10x15*, *30x40*, *25x60*. To find the center of mass we would do ((10 + 30 + 25) / 3 ~= 22) to find the horizontal component and ((15 + 40 + 60) / 3 ~= 39) for the vertical one. This means that our center of mass in this scenario is ***22x39***.

Given an image of *800x600* and a center of mass of *22x39*, let's find the left and top percentages:

From the left - $22/800 * 100 = 2.75\%$

From the top - $93/600 * 100 = 6.50\%$

Assuming we are cropping *300px* of the height, we'll crop top and bottom according to:

Top - $300 * 0.0275 = 9$

Bottom - just subtract top (*9px*) from the amount of crop (*300px*) - $300 - 9 = 291$

In an image of *800x600*, that means cropping from *(0, 9)* to *(800, 309)*, resulting in a *800x300px* image.

If we were cropping *300px* of the width instead, we would crop left and right according to:

Left - $300 * 0.065 = 20$

Right - just subtract left (*20px*) from the amount of crop (*300px*) - $300 - 20 = 280$

In an image of *800x600*, that means cropping from *(20, 0)* to *(520, 600)*, resulting in a *500x600px* image.

Available detectors

Face Detector

`thumbor.detectors.face_detector`

It detects faces, It considers the frontal part of the face for detection.

Feature Detector

`thumbor.detectors.feature_detector`

Detector used to find relevant focal points in the image. "Features" in this case and in other cases such as machine learning, are pieces of information (in this case, pieces of the image) that are relevant to solving a computational problem. For Thumbor we use this set of focal points to identify faces, for example. We use the first 10 set of points found.

Glasses Detector

`thumbor.detectors.glasses_detector`

It detects glasses on the faces.

Profile Detector

`thumbor.detectors.profile_detector`

It detects faces, It considers the side part of the face for detection.

Queued Detector

`thumbor.detectors.queued_detector`

Detector used to allow face detection process asynchronously.

Lazy Detection

Rationale

Thumbor performs pipeline detection of focal points for a given image. What this means is that it tries to determine one detection at a time, only skipping to the next if the current one fails.

We could configure it to run frontal face detection, then if it fails, try profile face detection and if it fails, best features detection.

As you can imagine, this is a cumbersome process and can take up precious cpu time from your server(s), eventually leading it to starvation of CPU. This is why we've implemented what we call Queued Detection.

Queued Detection

Configuring thumbor for lazy detecting is as simple as specifying a detector that supports queued detection.

Thumbor ships with three such detectors, called:

- `thumbor.detectors.queued_detector.queued_complete_detector`
- `thumbor.detectors.queued_detector.queued_face_detector`
- `thumbor.detectors.queued_detector.queued_feature_detector`

These are responsible, respectively, for pipeline detection of face and feature, only face or only feature.

You can check what additional configuration you need to add to your configuration file (thumbor.conf) in order to have the bundled detectors working.

How thumbor deals with queued detection?

When an image request arrives with a flag of "smart" detection, a call is made to the queued detector and it tells thumbor to skip smart detection and to serve the image with non-smart cropping (much faster).

The call to the queued detector places a message in a Redis Queue that will later be processed in order to detect focal points in the image.

The next time a request arrive for the same image and with a flag of "smart" detection, if information on detection is already available (if the message in the queue has already been processed), thumbor uses that info to do smart cropping and serves the result.

If the image still hasn't been processed, the same process from before applies, except thumbor won't place another message in the queue. This is intended as a way not to flood the queue with requests for the same image.

Redis Support

Thumbor supports Redis single node. and Redis sentinel.

2.4.4 Image loader

Pre-packaged loaders

thumbor comes pre-packaged with http and filesystem loaders.

Http loader

The http loader gets the original image portion of the URI and performs an HTTP GET to it. It then returns the image's string representation.

The http loader uses the **ALLOWED_SOURCES** configuration to determine whether or not an image is from a trusted source and can thus be loaded.

When `http_proxy`, `https_proxy` or `all_proxy` is set in the environment, the http loader uses the curl client, and libcurl applies those variables and `no_proxy` with curl's own rules. It ignores the uppercase `HTTP_PROXY`, and it fetches a target matching `no_proxy` directly even when `HTTP_LOADER_PROXY_HOST` and `HTTP_LOADER_PROXY_PORT` are set. Otherwise those two settings take precedence over the proxy variables. The curl client needs `pycurl`, which `pip install thumbor[all]` provides. Without it the loader logs a warning, ignores the proxy variables and fetches images directly.

`MAX_SOURCE_SIZE` remains present in the bundled legacy configuration, but the current HTTP loader does not consult it. Setting it does not limit downloaded bytes. Enforce a source-size limit in a reverse proxy or custom loader if your deployment requires one.

To use it you should set the **LOADER** configuration to **'thumbor.loaders.http_loader'**.

Https loader

The https loader works the same way as the http loader, except that it defaults to https instead of http.

To use it you should set the **LOADER** configuration to **'thumbor.loaders.https_loader'**.

Strict https loader

The strict https loader works the same way as the http loader, except that it only allows to load images over https.

To use it you should set the **LOADER** configuration to **'thumbor.loaders.strict_https_loader'**.

File loader

The file loader gets the original image portion of the URI and retrieves the file from the file system from a known path specified by the **FILE_LOADER_ROOT_PATH** configuration.

It joins the specified path with the configured root path and reads the image file if it exists.

To use it you should set the **LOADER** configuration to **'thumbor.loaders.file_loader'**.

File loader with http loader fallback

In some environments you need both kinds of file loading. For this use case you can use as loader with built-in fallback.

This loader will try to load images from local file storage. In case of an error the loader retry to load image with http_loader. If both attempts failed you'll get an error.

To use it you should set the **LOADER** configuration to **'thumbor.loaders.file_loader_http_fallback'**.

Compatibility Loader

The compatibility loader allows you to use legacy loaders (that do not support AsyncIO) in order to make it easier to transition to thumbor's Python 3 version.

To use it you should set the **LOADER** configuration to **'thumbor.compatibility.loader'**.

You also need to specify what's the legacy loader that the compatibility loader will use. Just set the **COMPATIBILITY_LEGACY_LOADER** configuration to the full name of the legacy loader you want to use. i.e.: COMPATIBILITY_LEGACY_LOADER = 'tc_aws.loaders.s3_loader'

2.4.5 Image storage

thumbor uses image storages to perform less retrievals of images from the sources, thus potentially saving expensive resources (such as outbound network).

Pre-Packaged Storages

thumbor comes with **filesystem** and a **mixed** storage. There's also a **nostorage** storage for debugging or benchmarking purposes.

Filesystem Storage

thumbor can store original images in the filesystem.

The file storage uses the **FILE_STORAGE_ROOT_PATH** configuration to save the images. It then joins the original image part of the URI to create the proper path to store the image in the filesystem.

There's a **STORAGE_EXPIRATION_SECONDS** option that will determine the time in seconds that a file is considered to be expired. When a file is expired, thumbor will try to retrieve the file using the specified *Image loader*. Expired files are not deleted; *Cleaning expired files* explains how to remove them.

To use the filesystem storage set the configuration option of **STORAGE** to **'thumbor.storages.file_storage'**.

NoStorage Storage

This is a storage intended for debugging or benchmarking purposes. It does not store any images and always returns None when thumbor asks for an image.

In order to use this storage set the configuration option of **STORAGE** to **'thumbor.storages.no_storage'**.

MixedStorage Storage

This is a storage intended for scenarios where you want to store the original images files one way and the security key another (or detector information).

A good example would be storing files in the filesystem, while storing security keys in a database.

In order to use this storage set the configuration option of **STORAGE** to **'thumbor.storages.mixed_storage'**.

You must specify the `MIXED_STORAGE_FILE_STORAGE`, `MIXED_STORAGE_CRYPTO_STORAGE` and `MIXED_STORAGE_DETECTOR_STORAGE` options to define the original images storage, the security key storage and the detector results storage, respectively. Here's a sample configuration:

```
MIXED_STORAGE_FILE_STORAGE = 'thumbor.storages.file_storage'
MIXED_STORAGE_CRYPTO_STORAGE = 'thumbor.storages.redis_storage'
MIXED_STORAGE_DETECTOR_STORAGE = 'thumbor.storages.redis_storage'
```

(continues on next page)

(continued from previous page)

```
FILE_STORAGE_ROOT_PATH = '/tmp/mypath'

REDIS_STORAGE_SERVER_HOST = 'localhost'
REDIS_STORAGE_SERVER_PORT = 6379
REDIS_STORAGE_SERVER_DB = 0
```

As you can see, you still have to tell thumbor the specific configurations for each storage you choose.

Compatibility Storage

The compatibility storage allows you to use legacy storages (that do not support AsyncIO) in order to make it easier to transition to thumbor's Python 3 version.

To use it you should set the **STORAGE** configuration to **'thumbor.compatibility.storage'**.

You also need to specify what's the legacy storage that the compatibility storage will use. Just set the **COMPATIBILITY_LEGACY_STORAGE** configuration to the full name of the legacy storage you want to use. i.e.: COMPATIBILITY_LEGACY_STORAGE = 'tc_aws.storages.s3_storage'

2.4.6 Result Storage

thumbor uses a result storage to improve the speed of responding subsequent requests for the same image.

When a request for a given image with a set of parameters arrive, thumbor processes the request and before returning it, asks for the result storage to store it.

The next time the same request arrives, it will get it from the result storage and return it, thus saving a lot of processing.

Pre-packaged result storages

thumbor comes pre-packaged with a filesystem result storage.

Filesystem

The file system result storage, as the name implies, stores images in the filesystem.

Images are stored in whatever path is specified in the `RESULT_STORAGE_FILE_STORAGE_ROOT_PATH`, and consequently retrieved from the same path.

By default, the file system result storage keeps images forever. You are allowed to specify an expiration, though, using the `RESULT_STORAGE_EXPIRATION_SECONDS` configuration. Again, as the name implies, it specifies the number of seconds with which files expire. Expired files are not deleted; *Cleaning expired files* explains how to remove them.

To use it you should set the `RESULT_STORAGE` configuration to `'thumbor.result_storages.file_storage'`.

Compatibility Result Storage

The compatibility result storage allows you to use legacy result storages (that do not support AsyncIO) in order to make it easier to transition to thumbor's Python 3 version.

To use it you should set the **RESULT_STORAGE** configuration to **'thumbor.compatibility.result_storage'**.

You also need to specify what's the legacy result storage that the compatibility result storage will use. Just set the **COMPATIBILITY_LEGACY_RESULT_STORAGE** configuration to the full name of the legacy result storage you want to use. i.e.: COMPATIBILITY_LEGACY_RESULT_STORAGE = 'tc_aws.result_storages.s3_storage'

2.4.7 Optimizers

Optimizers are utilities that will fine-tune some aspect of the result image thumbor generates.

Even though optimizers can change images in any way, the usual use cases for these are:

- Reduce image weight in bytes;
- Improve image quality.

Built-in Optimizers

jpegtran

Jpegtran is a lossless jpeg optimizer which can make your jpegs smaller by optimizing DCT coefficients. Information on jpegtran can be a bit difficult to find but the linux man page is pretty good: https://linux.die.net/man/1/jpegtran

Jpegtran can be used in conjunction with Thumbor. If the optimizer has been activated, Thumbor will first process your jpeg normally then it will hand the jpeg off to jpegtran for further optimizations before Thumbor returns the final image.

To use jpegtran with Thumbor you must first install jpegtran, various linux distros often provide a package by the same name or it can be installed from source. You should make sure that jpegtran is in PATH, do a `which jpegtran` and you should see an absolute path where the jpegtran resides. It is also possible to use mozjpeg's version of jpegtran as a drop-in replacement of libjpeg-turbo's version.

You also need to enable the Thumbor jpegtran optimizer in your thumbor.conf, like so:

```
OPTIMIZERS = [
  'thumbor.optimizers.jpegtran'
]
```

You can also manually specify the jpegtran path, like this:

```
JPEGTRAN_PATH = "/usr/local/bin/jpegtran"
```

Once activated, no extra url parameters are needed - jpegtran will run on all jpegs automatically. If you have opted to use progressive jpegs via the `PROGRESSIVE_JPEG` option, jpegtran will also honor and produce progressive jpegs.

It is possible to supply progressive scans file via `JPEGTRAN_SCANS_FILE` config option.

gifv

The gifv optimizer is able to convert gifs to mp4 or webm videos, often resulting in dramatically smaller sized files.

Gifv is categorized as experimental and should be used with caution. It uses ffmpeg to convert gifs to videos and so it's sensitive to changes with ffmpeg. It's recommended to lock your ffmpeg version with a fixed version (chef, docker, etc) and if updating make sure to check that the update doesn't break gifv. **FFmpeg version 3.2.4 is the current recommended version.** Later version, such as 3.3 will break the proper conversion of gif delays to frame durations in videos … meaning videos will not be the same length as equivalent gifs.

To enable gifv, ensure ffmpeg is in PATH and enable the optimizer in your config:

```
OPTIMIZERS = [
  'thumbor.optimizers.gifv',
]
```

Once activated, you must add the `gifv()` option to your filters list. An example request might look like this:

```
http://localhost:8888/unsafe/filters:gifv()/http://localhost/livingroom.gif
```

The above example will default to using the mp4 video container with h264 video. You can also be explicit:

```
http://localhost:8888/unsafe/filters:gifv(mp4)/http://localhost/livingroom.gif
```

or use explicitly specify webm

```
http://localhost:8888/unsafe/filters:gifv(webm)/http://localhost/livingroom.gif
```

Because videos (in mp4 or webm format) cannot contain alpha transparency a background color will be automatically added. The default color is white. You can also specify a background color:

```
http://localhost:8888/unsafe/filters:gifv():background_color(ff00ff)/http://localhost/
↪livingroom.gif
```

```
http://localhost:8888/unsafe/filters:gifv():background_color(f0f)/http://localhost/
↪livingroom.gif
```

```
http://localhost:8888/unsafe/filters:gifv():background_color(magenta)/http://localhost/
↪livingroom.gif
```

The color must be specified in 6 character hex, 3 character hex or color name. But 6 or 3 character hex are the preferred formats. Including a # symbol in your color will break the url if not url encoded and thumbor will error on the request. The recommendation is to not use them at all which also makes urls shorter. But if you must a leading `%23` will probably work.

2.5 Customizing Thumbor

2.5.1 Custom Storages

A storage persists original images and the metadata Thumbor associates with them. Configure it with `STORAGE`:

```
STORAGE = "my_package.storage"
```

Thumbor imports a class named `Storage` from that module and instantiates it with the current request context. Custom storages should inherit from `thumbor.storages.BaseStorage`.

The storage interface is asynchronous:

```
from thumbor.storages import BaseStorage


class Storage(BaseStorage):
    async def put(self, path, file_bytes):
        """Store the original image."""

    async def get(self, path):
        """Return the stored bytes, or None when the image is unavailable."""

    async def exists(self, path):
        """Return whether the original image is available."""
```

(continues on next page)

(continued from previous page)

```
    async def remove(self, path):
        """Remove the original image."""

    async def put_crypto(self, path):
        """Store the signing key associated with the image."""

    async def get_crypto(self, path):
        """Return the signing key associated with the image, or None."""

    async def put_detector_data(self, path, data):
        """Store detector data associated with the image."""

    async def get_detector_data(self, path):
        """Return detector data associated with the image, or None."""
```

Implement every operation used by the features you enable. Normal image loading first calls `exists` with the image path truncated to `MAX_ID_LENGTH` characters (32 by default, and 0 skips the check), then `get` and, after a miss, `put` followed by `put_crypto`. `put_crypto` is called even when `STORES_CRYPTO_KEY_FOR_EACH_IMAGE` is disabled, so it must exist at least as a no-op that returns `None`, as the built-in file storage does.

The `GET`, `HEAD` and `DELETE` routes of `/image/<id>` use `exists`, `get` and `remove` on this storage. Uploads through `POST /image` and `PUT /image/<id>` do not touch it: they only call `put` on the separate `UPLOAD_PHOTO_STORAGE` backend. The blacklist handlers use `exists`, `get` and `put`. Per-image signing reads the key back with `get_crypto`, and smart detection uses `get_detector_data` and `put_detector_data`.

The request context is available as `self.context`, including `self.context.config` and `self.context.server`. `self.context.request` is only set once the imaging handler builds the request parameters. It does not exist in the upload and blacklist handlers, or in the `exists` check the imaging handler runs before that, so derive storage keys from the `path` argument. See `thumbor/storages/file_storage.py` for the built-in reference implementation.

Storage methods run on Thumbor's event loop. Avoid blocking network or disk operations in an `async` method; use an asynchronous client or explicitly move blocking work off the event loop.

2.5.2 Custom Image Loaders

If thumbor's image loaders do not meet your needs, you can implement a custom loader as an importable Python module.

A loader module must expose an asynchronous `load(context, url)` function. `context` is the current request context, and `url` is the source-image path from the thumbor request. New loaders should return a `thumbor.loaders.LoaderResult`.

```
from thumbor.loaders import LoaderResult


async def load(context, url):
    image_bytes = context.config.MY_LOADER_IMAGES.get(url)

    if image_bytes is None:
        return LoaderResult(
            successful=False,
            error=LoaderResult.ERROR_NOT_FOUND,
        )

    return LoaderResult(buffer=image_bytes)
```

For compatibility, thumbor still accepts raw image bytes returned by old loaders, but new loaders should return `LoaderResult` so they can report errors and metadata explicitly.

A loader may also expose a synchronous `validate(context, url)` function. Thumbor calls it before loading the image. Return `True` to accept the URL and `False` to reject it.

```
def validate(context, url):
    return url.startswith("images/")
```

Configure the loader using its module name:

```
MY_LOADER_IMAGES = {
    "images/example.jpg": b"...",
}
LOADER = "mylib.loaders.custom_loader"
```

The HTTP loader at `thumbor/loaders/http_loader.py` and the filesystem loader at `thumbor/loaders/file_loader.py` are reference implementations.

2.5.3 Custom Result Storages

Thumbor imports a class named `Storage` from the module configured by `RESULT_STORAGE` and instantiates it with the current request context:

```
RESULT_STORAGE = "my_package.result_storage"
```

Custom result storages should inherit from `thumbor.result_storages.BaseStorage`. The current interface is:

```
from thumbor.result_storages import BaseStorage, ResultStorageResult


class Storage(BaseStorage):
    async def put(self, image_bytes):
        """Store the generated image for self.context.request."""

    async def get(self):
        """Return a ResultStorageResult, or None on a cache miss."""
```

`get` should return `None` when no valid cached result exists. On a cache hit, return `ResultStorageResult(buffer=..., metadata=...)`. Only `LastModified` affects the response. When `SEND_IF_MODIFIED_LAST_MODIFIED_HEADERS` is enabled it sets the `Last-Modified` header and answers `If-Modified-Since` with `304`, so it must be a timezone-aware `datetime`. The response content type is always detected from the buffer, so a `ContentType` entry does not change it, and `ContentLength` only feeds the `result_storage.bytes_read` metric.

`last_updated` is only consulted for compatibility when a backend returns raw bytes and conditional response headers are enabled. Thumbor awaits it, so it must be declared `async`. New implementations should return freshness information as `LastModified` metadata in `ResultStorageResult`.

`validate_path` and `normalize_path` are helpers used by the built-in file storage, not required methods for remote or key-value result storages. Whatever backend is used, derive the key from `self.context.request` and apply the automatic-format isolation rules below before either reading or writing.

Automatic image formats and cache keys

A result storage must keep responses for different automatic image-format capabilities in separate cache namespaces. This is required both for the individual `AUTO_*` options and for `AUTO_IMAGE_FORMAT_PREFERENCE`. Otherwise, clients requesting the same thumbor URL but advertising different formats can receive an incompatible cached image.

Use thumbor's cache-key helper when building the normalized path or key:

```
from thumbor.auto_image_format import get_auto_image_format_cache_key

format_key = get_auto_image_format_cache_key(
    self.context.config,
    self.context.request,
)
cache_namespace = format_key or "default"

# Include cache_namespace in the complete backend key used by both get() and
# put(). Do not use it as the key by itself.
```

Treat the returned value as an opaque discriminator. It represents the active automatic formats accepted by the request and can contain more than one format. For configurations that require isolation from legacy cache entries, it also contains an internal namespace version. In that mode the helper returns a discriminator even when no configured format was accepted. Include it in both read and write keys before any cache lookup.

If a custom handler overrides `BaseHandler.accepts_mime_type` while `AUTO_AVIF`, `AUTO_HEIF`, `AUTO_JPG`, `AUTO_PNG` or a preference containing one of those formats is active, thumbor bypasses result storage for that request. The override may depend on the image engine, filters or other state that does not exist during the pre-load cache lookup, so no safe cache discriminator can be calculated at that point. The override continues to run during output-format selection, at the same stage as before. The same bypass applies to request objects built by custom handler code without the parsed `accepts_*` attributes: their format decisions fall back to reading the Accept header, so no cache key computed from the missing attributes could be trusted. `AUTO_WEBP`-only and `AUTO_PNG_TO_JPG`-only configurations are unaffected because this hook has never controlled those conversions.

When enabling or reordering `AUTO_IMAGE_FORMAT_PREFERENCE`, an entry from an older `default` or `auto_webp` namespace is not necessarily valid for the new namespace. Treat such an entry as a cache miss and regenerate it. Do not copy or move legacy objects into a new format namespace unless their actual content type has been verified. Existing custom storages that do not use the helper must adopt it before enabling these automatic formats. Purge their generated-image cache once during that rollout; purging without first isolating the new keys does not prevent future format collisions.

2.5.4 Custom Filters

Filters transform images as part of Thumbor's processing pipeline. A filter module must expose a class named `Filter` that inherits from `thumbor.filters.BaseFilter`. Filter methods are asynchronous and must be decorated with `filter_method`.

The example below implements `quality(99)`, a filter that takes a positive number and sets the image quality to that value:

> **Note**
>
> Yep, this filter already exists and is built-in, but it is simple enough that we can talk about how to do it. Let's get on with it.

```
from thumbor.filters import BaseFilter, filter_method


class Filter(BaseFilter):
    @filter_method(BaseFilter.PositiveNumber)
    async def quality(self, value):
        self.context.request.quality = value
```

Let's analyse it:

- The `filter_method` decorator takes as parameters any number of types (more on types below) you want to have as arguments to your filter;
- The filter method should be named according to how you want it to be invoked by thumbor (a.k.a the URL part). In our example, our filter will be invoked with `quality(99)`;
- The filter method is just an async function that you can do whatever you need with the image.

The `Filter` instance receives the current request context as `self.context`. The current image engine is available as `self.engine`.

When a metrics backend is enabled, each filter execution emits a `filter.<filter_name>.time` timing. Successful executions also emit `filter.<filter_name>.count`, while failed executions emit `filter.<filter_name>.error`. The `<filter_name>` portion comes from the decorated method name, so it should stay stable if you rely on dashboards or alerts.

And that's it, we got our filter. In order to use it, we need to put it in our `thumbor.conf`:

```
from thumbor.filters import BUILTIN_FILTERS

FILTERS = BUILTIN_FILTERS + [
    "mylib.filters.quality",
]
```

Each entry in `FILTERS` is the full name of a module, not the full name of the class. Thumbor imports `Filter` from each configured module in list order and indexes the classes by filter name, so when two modules define the same filter name the last entry wins. List order does not affect execution order. Filters are grouped by their `phase` (pre-load, after-load or the default post-transform), each group runs at its own point of the pipeline, and filters in the same group run in the order they appear in the request URL. A module that raises `ImportError` is logged as a warning and skipped. Any other error while importing it, or a module without a `Filter` class, stops startup.

Available Filter Argument Types

Each parameter type has a regular expression that matches arguments of the given type, as well as a python type.

For more details on each type, see `thumbor/filters/__init__.py` in the thumbor repository.

- `BaseFilter.PositiveNumber`;
- `BaseFilter.PositiveNonZeroNumber`;
- `BaseFilter.NegativeNumber`;
- `BaseFilter.Number`;
- `BaseFilter.DecimalNumber`;
- `BaseFilter.Boolean`;
- `BaseFilter.String`.

2.5.5 Custom Engines

Thumbor imports a class named `Engine` from the module configured by `ENGINE` and creates an instance with the current request context:

```
ENGINE = "my_package.engine"
```

Start by subclassing `thumbor.engines.BaseEngine`:

```
from thumbor.engines import BaseEngine


class Engine(BaseEngine):
    def create_image(self, buffer):
        """Decode buffer and return the backend image object."""

    @property
    def size(self):
        """Return the current (width, height)."""

    def crop(self, left, top, right, bottom):
        """Crop using Thumbor's pixel coordinates."""

    def resize(self, width, height):
        """Resize the current image."""

    def flip_horizontally(self):
        """Flip the current image horizontally."""

    def flip_vertically(self):
        """Flip the current image vertically."""

    def read(self, extension=None, quality=None):
        """Encode and return the final image bytes."""
```

Keep the defaults on `read()`. `BaseEngine` declares both arguments as required, but the `frame` filter calls `read()` without arguments on a new instance of the configured engine, as the PIL engine allows.

`BaseEngine.load()` detects the extension and converts SVG input before calling `create_image()`. After `create_image()` returns, it reads EXIF metadata from `self.exif` and wraps animated input in per-frame engines. Both steps depend on `create_image()`: set `self.exif` to the raw EXIF bytes when the format carries them, and return a list or tuple of frames for animated input so that `ALLOW_ANIMATED_GIFS` handling applies (`ALLOW_ANIMATED_WEBP` also enables it for WebP input). Reuse `load()` unless your format requires a different loading lifecycle.

A production engine must also implement the operations used by the filters and features it supports. These can include `gen_image`, `rotate`, `image_data_as_rgb`, `get_image_data`, `set_image_data`, `get_image_mode`, `paste`, `enable_alpha`, `convert_to_grayscale`, `draw_rectangle`, `extract_cover`, `has_transparency`, `avif_enabled`, `heif_enabled` and `read_multiple`. Unsupported optional operations should fail explicitly or be paired with configuration that prevents the corresponding feature from being selected.

The active request and configuration are available through `self.context`. Thumbor creates a new engine for every request and does not call `cleanup()` when the request ends; it only runs for the server context's engine at shutdown. Release native or temporary resources inside the operations that allocate them instead of relying on `cleanup()`.

Use `thumbor.engines.pil.Engine` as the reference implementation. Test at least load/read round trips, crop, resize, orientation, transparency, malformed input, animated input if supported, and every automatic output format the engine

advertises.

2.5.6 Custom detection

If you need more detection than the pre-packaged detectors are able to give you (i.e.: you need to detect glasses), you can always implement your own detectors.

If your detector can be found using python's import mechanism, thumbor will be able to use it. Just add its full name to the detectors *Configuration*.

Creating a Custom Detector

The face detector in the thumbor repository demonstrates how easy it is to implement your own custom detector. Its source lives at `thumbor/detectors/face_detector/__init__.py`.

A detector module must expose a class named `Detector` that inherits from `thumbor.detectors.BaseDetector`. Thumbor constructs it with the current request context, its position in the configured detector list, and the complete detector list.

Implement an asynchronous `detect(self)` method. Focal points found by the detector should be appended to `self.context.request.focal_points`. If the detector finds no points, call `await self.next()` so the next configured detector can run.

`detect` should return `None`, as the example below does. Thumbor only reads the return value of the first detector. A non-`None` value is treated as a list of focal-point dictionaries and each item is passed to `FocalPoint.from_dict`, so returning `FocalPoint` objects fails. The return values of detectors reached through `self.next()` are discarded.

```
from thumbor.detectors import BaseDetector
from thumbor.point import FocalPoint


class Detector(BaseDetector):
    async def detect(self):
        # Replace this with the detector's own image-analysis result.
        detected_rectangles = []

        if not detected_rectangles:
            await self.next()
            return

        for left, top, width, height in detected_rectangles:
            self.context.request.focal_points.append(
                FocalPoint.from_square(
                    left,
                    top,
                    width,
                    height,
                    origin="my-detector",
                )
            )
```

Add the module, not the class name, to `DETECTORS`:

```
DETECTORS = [
    "mylib.detectors.my_detector",
]
```

Detector order is significant. Thumbor instantiates and calls only the first detector in this list; `await self.next()` constructs and invokes the next one. Focal points produced by a detection run are cached in the configured `STORAGE` with `put_detector_data`, keyed by the image URL, and later requests read them back with `get_detector_data` instead of running the detectors.

2.5.7 Custom Image Optimizers

An optimizer receives the bytes produced by the engine and may return an optimized replacement. Add optimizer modules to `OPTIMIZERS` in execution order:

```
OPTIMIZERS = [
    "my_package.optimizer",
]
```

Each module must expose a class named `Optimizer`. Subclass `thumbor.optimizers.BaseOptimizer`:

```
import shutil

from thumbor.optimizers import BaseOptimizer


class Optimizer(BaseOptimizer):
    def should_run(self, image_extension, image_buffer):
        return image_extension in {".jpg", ".jpeg"}

    def optimize(self, image_buffer, input_file, output_file):
        # input_file and output_file are paths. Replace the copy with the
        # command that writes the optimized image to output_file.
        shutil.copyfile(input_file, output_file)
```

The base `run_optimizer()` implementation checks `should_run()`, writes the buffer to a temporary input file, calls `optimize()` with the input and output file paths and returns the bytes written to the output file. The return value of `optimize()` is ignored, so it must always write `output_file`: an empty output file makes the request fail. Override `run_optimizer(image_extension, buffer)` instead when the optimizer works in memory or needs a different process lifecycle.

An overridden `run_optimizer()` should return the original buffer when optimization is skipped or fails safely. A non-`None` return value becomes the input to the next configured optimizer, so the order in `OPTIMIZERS` is significant. Thumbor instantiates each optimizer with the request context, available as `self.context`.

Optimizers run synchronously while the response is being produced. Bound external-process execution time and avoid leaving child processes or temporary files behind. See `thumbor.optimizers.jpegtran.Optimizer` for an in-memory `run_optimizer()` implementation.

2.5.8 Custom Error Handlers

A custom error-handler module must expose a class named `ErrorHandler`. Thumbor constructs the class once at startup and passes the loaded configuration to its constructor. `handle_error` is synchronous and is called with the current request context, the Tornado request handler, and exception information.

```
import logging


logger = logging.getLogger(__name__)
```

(continues on next page)

(continued from previous page)

```
class ErrorHandler:
    def __init__(self, config):
        self.config = config

    def handle_error(self, context, handler, exception):
        # Thumbor normally passes the tuple returned by sys.exc_info().
        if isinstance(exception, tuple):
            exc_info = exception
        else:
            exc_info = (
                type(exception),
                exception,
                exception.__traceback__,
            )

        logger.error(
            "Error while handling %s",
            handler.request.uri,
            exc_info=exc_info,
        )
```

`context` is thumbor's context for the current request, `handler` is the Tornado request handler, and `exception` is normally the `(type, value, traceback)` tuple returned by `sys.exc_info()`. Thumbor passes all three as keyword arguments, so the parameter names `context`, `handler` and `exception` are part of the contract. The method is not awaited, so it must be a regular synchronous method.

Configure the module name, not the class name, and make sure it is importable from thumbor's Python environment:

```
USE_CUSTOM_ERROR_HANDLING = True
ERROR_HANDLER_MODULE = "mylib.error_handling"
```

2.5.9 Custom Handler Lists

Handler Lists are responsible for adding new handlers to thumbor.

Even thumbor's own handlers (other than the default image crop handler) are added using handler lists(healthcheck, blacklist...).

Built-in Handler Lists

Thumbor comes with three handler lists built-in:

- `thumbor.handler_lists.healthcheck`;
- `thumbor.handler_lists.upload`;
- `thumbor.handler_lists.blacklist`.

The healthcheck handler list adds a handler at whatever is in the `HEALTHCHECK_ROUTE` config.

The blacklist handler list adds a `/blacklist` handler that can be used to blacklist images.

The upload handler list adds two handlers for uploading and retrieving uploaded images.

Writing a new Handler List

A handler list is an importable Python module with a synchronous `get_handlers(context)` function. The function receives thumbor's application context and returns a list of Tornado handler specifications.

The third item in a handler specification is passed to the handler's `initialize` method. Handlers that inherit from `ContextHandler` should receive thumbor's context using `{"context": context}`.

```
from typing import Any

from thumbor.handler_lists import HandlerList
from thumbor.handlers import ContextHandler


class IndexHandler(ContextHandler):
    async def get(self):
        self.write("Hello from my handler")


def get_handlers(context: Any) -> HandlerList:
    if not context.config.get("SOMETHING_ENABLED", False):
        return []

    return [
        (r"/my-url/?", IndexHandler, {"context": context}),
    ]
```

After your handler list can be imported with python (check with `python -c 'import <<your handler list module>>'`), just add it to thumbor's config:

```
from thumbor.handler_lists import BUILTIN_HANDLERS

# Two things worth noticing here:
# 1) Handler list order indicates precedence. The first match is executed.
# 2) Include thumbor's built-ins to preserve thumbor functionality.
HANDLER_LISTS = BUILTIN_HANDLERS + [
    "my.handler_list",
]
```

Handler lists are evaluated in configuration order, and the first matching route is used. The built-in imaging route is appended after every configured handler list. Keep `BUILTIN_HANDLERS` unless you intentionally want to remove thumbor's healthcheck, upload, and blacklist routes.

2.5.10 Plugins

With its pluggable architecture, Thumbor provides extension points for storages, result storages, loaders, detectors, filters, engines, optimizers, metrics, error handlers, URL signers and handler lists.

If your plug-in is not listed here, please create an issue with the details and we'll add it here.

Important

The projects listed here are maintained independently and may target different Thumbor or Python versions. Before deploying one, verify its supported versions, release status and configuration in the plugin's own documentation. A listing on this page is not a compatibility guarantee.

Thumbor configuration values name importable Python modules. Depending on the extension point, the module must expose the class expected by Thumbor: `Storage`, `Engine`, `Optimizer`, `Detector`, `Filter`, `Metrics`, `UrlSigner` or `ErrorHandler`. Loader and handler-list entries are module-based interfaces. See the corresponding customization guide for current method signatures.

Plugins may define configuration keys that are not part of Thumbor core. Those keys are available to the plugin through `context.config`; keep their spelling and Python value types exactly as documented by the plugin. Environment variable overrides are always strings: moving a list, boolean, number, dictionary or `None` from `thumbor.conf` to an environment variable changes its type.

Defining plugin configuration

Plugin modules should register their defaults through `thumbor.config.Config` when they are imported:

```
from thumbor.config import Config


Config.define(
    "MY_PLUGIN_TIMEOUT",
    5,
    "Maximum time in seconds for my plugin operation",
    "My Plugin",
)
```

Import `Config` from `thumbor.config`, which re-exports `derpconf.config.Config`. Its definition registry stays live, so a setting registered while a configured plugin is imported becomes available to the already loaded config. An explicit value from `thumbor.conf` continues to take precedence over the plugin default:

```
MY_PLUGIN_TIMEOUT = 10
LOADER = "my_plugin.loader"
```

Any uppercase name assigned in `thumbor.conf` is kept on the loaded configuration even when it has no registered default. Registration is still recommended because it supplies a default, description and group metadata. Plugin code can read the resolved value through the configuration object it receives directly or through `context.config`.

Storages

thumbor_aws

By Thumbor Community.

Thumbor is a smart imaging service. It enables on-demand crop, resizing and flipping of images.

AWS is a cloud service, providing - among other things - storage capabilities.

This module provides support for AWS S3 interconnection, as a loader, a storage and/or a result storage.

- *URL:* https://github.com/thumbor-community/aws
- *Installing:* `pip install tc_aws`

For complete, version-specific configuration, use the documentation in the plugin repository linked above.

thumbor_hbase

By Damien Hardy.

Thumbor is a smart imaging service. It enables on-demand crop, resizing and flipping of images.

Hbase is a column oriented database from the hadoop ecosystem.

This module provide support for Hadoop Hbase as large auto replicant key/value backend storage for images in Thumbor.

- *URL:* https://github.com/dhardy92/thumbor_hbase
- *Installing:* `pip install thumbor_hbase`

Using it is simple, just change your configuration in thumbor.conf:

```
HBASE_STORAGE_SERVER_HOST = "localhost"
HBASE_STORAGE_SERVER_PORT = 9000
HBASE_STORAGE_TABLE = "storage-table"
HBASE_STORAGE_FAMILY = "storage-family"
```

If you want to use thumbor_hbase for loading original images, change your thumbor.conf to read:

```
LOADER = "thumbor_hbase.loader"
```

If you want to use thumbor_hbase for storage of original images, change your thumbor.conf to read:

```
STORAGE = "thumbor_hbase.storage"
```

thumbor_mongodb

By Damien Hardy.

Thumbor is a smart imaging service. It enables on-demand crop, resizing and flipping of images.

MongoDB is a document oriented NoSQL database.

This plugin for Thumbor is a loader that can reach images from a mongodb collection based on its Object(_id).

- *URL:* https://github.com/dhardy92/thumbor_mongodb
- *Installing:* `pip install thumbor_mongodb`

Using it is simple, just change your configuration in thumbor.conf:

```
LOADER = 'thumbor_mongodb.loader'
MONGO_LOADER_CNX_STRING = 'mongodb://mongodbserver01,mongodbserver02:27017'
MONGO_LOADER_SERVER_DB = 'thumbor'
MONGO_LOADER_SERVER_COLLECTION = 'images'
MONGO_LOADER_DOC_FIELD = 'content'
```

thumbor_riak

By Damien Hardy.

Riak is a distributed document oriented database implementing the consistent hashing algorithm from the Dynanmo publication by Amazon.

This module provide support for Riak as a large auto replicant key/value backend storage for images in Thumbor.

- *URL:* https://github.com/dhardy92/thumbor_riak
- *Installing:* `pip install thumbor_riak` (require thumbor)

Using it is simple, just change your configuration in thumbor.conf:

```
# Use riak for storage.
STORAGE = 'thumbor_riak.storage'

# Put the url for your riak install here
RIAK_STORAGE_BASEURL = "http://my-riak-install-base-url"
```

thumbor_rackspace

By David Mann.

This plugin allows users to store objects in the Rackspace cloud for result storage.

- *URL:* https://github.com/CodingNinja/thumbor_rackspace
- *Installing:* `pip install thumbor_rackspace`

Using it is simple, just change your configuration in thumbor.conf:

```
# Use rackspace for result storage.
# For more info on result storage: https://github.com/thumbor/thumbor/wiki/Result-storage
RESULT_STORAGE = 'thumbor_rackspace.result_storages.cloudfile_storage'

# Pyrax Rackspace configuration file location
RACKSPACE_PYRAX_CFG = "/var/thumbor/.pyrax.cfg"

# Result Storage options
RACKSPACE_RESULT_STORAGE_EXPIRES = True # Set TTL on cloudfile objects
RACKSPACE_RESULT_STORAGES_CONTAINER = "cloudfile-container-name"
RACKSPACE_RESULT_STORAGES_CONTAINER_ROOT = "/"
```

thumbor_ceph

By Laurent Barbe.

Ceph a distributed object store designed to provide excellent performance, reliability and scalability.

This module provide support for Ceph RADOS as backend storage for images.

- *URL:* https://github.com/ksperis/thumbor_ceph
- *Installing:* `apt-get install python-ceph && pip install thumbor_ceph`

Configuration in thumbor.conf:

```
################################## File Storage ###################################
STORAGE = 'thumbor_ceph.storages.ceph_storage'
CEPH_STORAGE_POOL = 'thumbor'

################################### Upload ########################################
UPLOAD_PHOTO_STORAGE = 'thumbor_ceph.storages.ceph_storage'

################################ Result Storage ###################################
RESULT_STORAGE = 'thumbor_ceph.result_storages.ceph_storage'
CEPH_RESULT_STORAGE_POOL = 'thumbor'
```

For monitors and keys, the values used are those defined in the configuration file ceph.conf.

thumbor_spaces

By Siddhartha Mukherjee.

This plugin allows users to store objects in the DigitalOcean Spaces for result storage.

- *URL:* https://github.com/siddhartham/thumbor_spaces
- *Installing:* `pip install thumbor_spaces`

Using it is simple, just change your configuration in thumbor.conf:

```
# Use DigitalOcean Spaces for result storage.
# For more info on result storage: https://github.com/thumbor/thumbor/wiki/Result-storage
RESULT_STORAGE = 'thumbor_spaces.result_storages.spaces_storage'

SPACES_REGION='xxx'

SPACES_ENDPOINT='xxx'

SPACES_KEY='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'

SPACES_SECRET='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'

SPACES_BUCKET='your-bucket-name'
```

Metrics

thumbor_prometheus

By Simon Effenberg.

Prometheus a monitoring and alerting toolkit.

This module provide support for Prometheus as metrics collector.

- *URL:* https://github.com/thumbor-community/prometheus
- *Installing:* `pip install tc_prometheus`

Configuration in thumbor.conf:

```
############################### Extensibility #################################
METRICS = 'tc_prometheus.metrics.prometheus_metrics'

# optional with defaults
PROMETHEUS_SCRAPE_PORT = 8000 # Port the prometheus client should listen on
```

Extensions

thumborshortener

By Thumbor Community.

Thumbor is a smart imaging service. It enables on-demand crop, resizing and flipping of images.

This module provides URL shortening capabilities for Thumbor. It will create an API that can shorten a thumbor URL, and then routing capabilities to reroute the shortened URL to the correct image.

The shortened URL / real URL mapping is stored within redis.

- *URL:* https://github.com/thumbor-community/shortener
- *Installing:* `pip install tc_shortener`

To get exhaustive details about configuration options & setting it up, go to the plugin documentation.

Engines

thumbor-video-engine

By The Atlantic.

This engine extends thumbor so that it can read, crop, and transcode audio-less video files using FFmpeg. It supports input and output of animated GIF, animated WebP, WebM (VP9) video, and MP4 (H.264 and H.265).

- *URL:* https://github.com/theatlantic/thumbor-video-engine
- *Installing:* `pip install thumbor-video-engine`

Configuration in thumbor.conf:

```
ENGINE = 'thumbor_video_engine.engines.video'
FILTERS = [
    # Enables transcoding between video formats (in addition to the image
    # formats already supported by thumbor.filters.format)
    'thumbor_video_engine.filters.format',
    # Allows outputting a still frame from a video as an image
    'thumbor_video_engine.filters.still',
]

# optional, if you are already using a custom image engine
IMAGING_ENGINE = 'opencv_engine'
```

For a full list of configuration options and filters, read the project's documentation.

2.5.11 Libraries

Even though the process of generating safe image URLs is explained in the *Security* page, we'll try to provide libraries in each programming language to ease this process.

Available Libraries

Python

- libthumbor - Python's extensions to thumbor. These are used to generate safe urls among others.
- django-thumbor - A django app with templatetags for resizing images with thumbor (by ricobl).
- django-thumborstorage - A Django custom storage for Thumbor backend (by Stanislas Guerra).

Node.js

- thumbor-client - Thumbor client for Node.js and Browser, SSR friendly (by azabroflovski).
- ThumborJS - Javascript's extension to thumbor. These are used to generate safe urls, encrypted urls among others (by Rafael Carício).
- ThumborUrlBuilder - Thumbor client for Node JS (by David Caramelo).
- thumbor - Thumbor client for Node JS (by PolicyMic).

Ruby

- ruby-thumbor - Ruby's gem to interact with thumbor server.
- thumbor_rails - Ruby's gem to make easier to generate urls in Rails projects.

Java

- Pollexor - Java client for the Thumbor image service which allows you to build URIs in an expressive fashion using a fluent API.
- Enterprise edition - Java library for encrypted URLs. Deprecated in favor of Pollexor.

PHP

- Thumbor-PHP - PHP implementation of URL generator for Thumbor. It also supports Laravel Framework.
- Phumbor - A minimal PHP client for generating Thumbor URLs.
- Phumbor for Laravel - A Laravel package providing a facade for Phumbor.
- Phumbor for Symfony2 - A Symfony2 Bundle providing a facade for Phumbor.

Swift

- Bumbo - A swifty client for Thumbor

Objective-C

- OCThumbor - Objective-C for the Thumbor image service which allows you to build URIs in an expressive fashion using a fluent API.

.NET

- DotNetThumbor - DotNet client for the Thumbor image service. Provides an expressive fluent API.

Delphi

- DelphiThumbor - Delphi class to thumbor. These are used to generate safe urls among others (by Marlon Nardi).

Implementing a library

If you want to provide a library to enable easy usage of thumbor in your favorite programming language, please send an e-mail to **thumbor@googlegroups.com** and we'll add it here.

Below are all the scenarios we think are worth testing automatically so you can guarantee compatibility with thumbor. Please note that this is not meant to be a replacement for TDD or for any other testing methodology you might want to use. These are just helper scenarios that we thought would help any library developers.

Library Tests - Generating HMAC of the URLs

We sincerely advise you to have thumbor installed in your machine, so you can implement a method in your tests that has thumbor generate a signature for your URL so you can compare with your own signature. This way you can make sure your url formatting and signing are working properly.

Here's how it was implemented in Ruby:

```
def sign_in_thumbor(key, str)
    #bash command to call thumbor's decrypt method
    command = "python3 -c 'from libthumbor.url_signers.base64_hmac_sha1 import UrlSigner;
↪ signer = UrlSigner(\"" << key << "\"); print(signer.signature(\"" << str << "\").
↪decode(\"utf-8\"))'"

    #execute it in the shell using ruby's popen mechanism
    result = Array.new
    IO.popen(command) { |f| result.push(f.gets) }

    result.join('')
end
```

You should be able to implement this easily in any modern programming language. It makes for very reliable tests.

Library Tests - Scenarios

Remember that these are in pseudo-code (BDD-like) language, and not in any programming language specifically.

Encryption Testing

These scenarios assume that you separate the logic of composing the url to be signed into a different "module", that is to be tested with the URL Testing Scenarios after these scenarios.

Scenario 1 - Signing of a known url results

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And a width of 300
    And a height of 200
When
    I ask my library for a signed url
Then
    I get '/8ammJH8D-7tXy6kU3lTvoXlhu4o=/300x200/my.server.com/some/path/to/image.jpg'␣
↪as url
```

Scenario 2 - Thumbor matching of signature with my library signature

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And a width of 300
    And a height of 200
When
    I ask my library for an encrypted URL
Then
    I get the proper url (/8ammJH8D-7tXy6kU3lTvoXlhu4o=/300x200/my.server.com/some/path/
↪to/image.jpg)
```

Scenario 3: metadata signature

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And the meta flag
When
    I ask my library for an encrypted URL
Then
    I get the proper url (/Ps3ORJDqxlSQ8y00T29GdNAh2CY=/meta/my.server.com/some/path/to/
↪image.jpg)
```

Scenario 4: smart-crop signature

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And the smart flag
When
    I ask my library for an encrypted URL
Then
    I get the proper url (/-2NHpejRK2CyPAm61FigfQgJBxw=/smart/my.server.com/some/path/to/
↪image.jpg)
```

Scenario 5: fit-in signature

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And the fit-in flag
When
    I ask my library for an encrypted URL
Then
    I get the proper url (/uvLnA6TJlF-Cc-L8z9pEtfasO3s=/fit-in/my.server.com/some/path/
↪to/image.jpg)
```

Scenario 6: filter signature

```
Given
    A security key of 'my-security-key'
    And an image URL of "my.server.com/some/path/to/image.jpg"
    And a 'quality(20)' filter
    And a 'brightness(10)' filter
When
    I ask my library for an encrypted URL
Then
    I get the proper url (/ZZtPCw-BLYN1g42Kh8xTcRs0Qls=/
↪filters:brightness(10):contrast(20)/my.server.com/some/path/to/image.jpg)
```

You should test the same kind of tests for horizontal and vertical flip, horizontal and vertical alignment and manual cropping.

More Information

- *Security*

2.6 Administration

2.6.1 Configuration

thumbor's configuration file is just a regular python script that gets loaded by thumbor.

When no configuration path is provided, thumbor looks for `thumbor.conf` in this order:

1. the current working directory;
2. the current user's home directory;
3. `/etc`;
4. the installed `thumbor` package directory.

The first file found is loaded. Pass `-c` or `--conf` to load a specific file instead. An explicit path is used as provided; thumbor does not expand ~ in that argument. Relative explicit paths are resolved from the current working directory.

To generate a commented template for the installed version, run:

```
thumbor-config > ./thumbor.conf
```

This command writes to standard output. Shell redirection creates or replaces the destination file; it does not merge with an existing configuration.

Override configuration through environment variables

Environment overrides are disabled by default. To enable the legacy derpconf environment lookup, pass a value to `--use-environment`, for example:

```
SECURITY_KEY=my-secret-key thumbor --use-environment=true
```

The environment variable name must exactly match the configuration key; there is no prefix. Environment values are returned as strings and override values loaded from `thumbor.conf` whenever that key is read. Consequently, this mode is safe only for string-valued settings. Boolean, integer, list, dictionary and other typed settings should remain in `thumbor.conf`.

`--use-environment` currently requires a value. A bare `--use-environment` argument is not accepted. Omit the option to leave overrides disabled; `--use-environment=False` also enables them because the current CLI treats its value as a nonempty string.

For boolean settings, `UPLOAD_ENABLED=True` enables uploads because `"True"` is a nonempty string. `UPLOAD_ENABLED=False` also enables them, even when `thumbor.conf` leaves them disabled, because `"False"` is a nonempty string too. Likewise, `QUALITY=85` stays the string `"85"`, which the JPEG encoder rejects, so JPEG output fails instead of using quality 85.

Converting environment values

The Python configuration file can read and convert environment variables itself. For example, save this as `thumbor.conf`:

```
import os
```

(continues on next page)

(continued from previous page)

```
def env_bool(name):
    return os.environ.get(name, "false").strip().lower() == "true"


SECURITY_KEY = os.environ["SECURITY_KEY"]
UPLOAD_ENABLED = env_bool("UPLOAD_ENABLED")
UPLOAD_DELETE_ALLOWED = env_bool("UPLOAD_DELETE_ALLOWED")
UPLOAD_PUT_ALLOWED = env_bool("UPLOAD_PUT_ALLOWED")
QUALITY = int(os.environ.get("QUALITY", "80"))
```

This helper enables a boolean option only for `true`, ignoring case and surrounding whitespace. Missing variables and other values leave it disabled. `SECURITY_KEY` is read without a default, so startup fails when the variable is missing instead of keeping the insecure default key. Set other configuration options in the file as usual.

Load the file without `--use-environment`:

```
thumbor -c ./thumbor.conf
```

The file still reads the environment through `os.environ`. Enabling direct environment overrides would replace its converted values with strings again. See *Docker Compose configuration* for complete Compose examples.

Extensibility Section

LOADER

The loader is responsible for retrieving the source image that thumbor will work with. This configuration defines the module that thumbor will use for it. **This must be a full namespace module (a.k.a. python has to be able to *import* it).**

```
LOADER = 'thumbor.loaders.http_loader'
```

STORAGE

The storage is responsible for storing the source image bytes and related metadata (face-detection, encryption and such) so that we don't keep loading it every time. **This must be a full namespace module (a.k.a. python has to be able to *import* it).**

```
STORAGE = 'thumbor.storages.file_storage'
```

MIXED_STORAGE_FILE_STORAGE

If you are using thumbor's mixed storage (`thumbor.storages.mixed_storage`), this is where you specify the storage that will be used to store images. **This must be a full namespace module (a.k.a. python has to be able to *import* it).**

```
MIXED_STORAGE_FILE_STORAGE = 'thumbor.storages.file_storage'
```

MIXED_STORAGE_CRYPTO_STORAGE

If you are using thumbor's mixed storage (`thumbor.storages.mixed_storage`), this is where you specify the storage that will be used to store cryptography information. **This must be a full namespace module (a.k.a. python has to be able to *import* it).**

```
MIXED_STORAGE_CRYPTO_STORAGE = 'thumbor.storages.file_storage'
```

MIXED_STORAGE_DETECTOR_STORAGE

If you are using thumbor's mixed storage (`thumbor.storages.mixed_storage`), this is where you specify the storage that will be used to store facial and feature detection results. **This must be a full namespace module (a.k.a. python has to be able to *import* it)**.

```
MIXED_STORAGE_DETECTOR_STORAGE = 'thumbor.storages.file_storage'
```

RESULT_STORAGE

The result storage is responsible for storing the resulting image with the specified parameters (think of it as a cache), so that we don't keep processing it every time a request comes in. **This must be a full namespace module (a.k.a. python has to be able to *import* it).**

```
RESULT_STORAGE = 'thumbor.result_storages.file_storage'
```

ENGINE

The engine is responsible for transforming the image. **This must be a full namespace module (a.k.a. python has to be able to *import* it).**

Currently, thumbor ships with only the `thumbor.engines.pil` imaging engine. A few years ago we conducted a comparison between the engines and there was no clear winner. Given PIL was the engine we were using at the time, we decided to stick with it. Other open source engines exist and you can find more about them in the plug-in section of the docs.

```
ENGINE = 'thumbor.engines.pil'
```

URL_SIGNER

The url signer is responsible for validation and signing of requests to prevent url tampering, which could lead to denial of service (example: filling the result_storage by specifying a different size). **This must be a full namespace module (a.k.a. python has to be able to *import* it).**

```
URL_SIGNER = 'libthumbor.url_signers.base64_hmac_sha1'
```

Filters Section

In order to specify the filters that thumbor will use, you need a configuration key called `FILTERS`. This is a regular python list with the full names (names that python can import) of the filter modules you want to use.

i.e.:

```
FILTERS = [
    'thumbor.filters.brightness',
    'thumbor.filters.contrast',
    'thumbor.filters.rgb',
    'thumbor.filters.round_corner',
    'thumbor.filters.quality',
    'thumbor.filters.noise',
```

(continues on next page)

(continued from previous page)

```
    'thumbor.filters.watermark',
]
```

Metadata Section

META_CALLBACK_NAME

If you want thumbor to use JSONP for image metadata instead of using JSON, just set this variable to the callback name you want.

```
META_CALLBACK_NAME = 'thumbor_callback'  # Or None for no callback
```

Face and Feature Detection Section

DETECTORS

This options specifies the detectors that should run the image to check for focal points.

i.e.:

```
 DETECTORS = [
   'thumbor.detectors.face_detector',
   'thumbor.detectors.feature_detector'
]
```

Cascade Files

This option specifies the cascade (XML) file paths to train openCV to find faces or other objects.

```
## The cascade file that opencv will use to detect faces.
FACE_DETECTOR_CASCADE_FILE = 'haarcascade_frontalface_alt.xml'

## The cascade file that opencv will use to detect glasses.
GLASSES_DETECTOR_CASCADE_FILE = 'haarcascade_eye_tree_eyeglasses.xml'

## The cascade file that opencv will use to detect profile faces.
PROFILE_DETECTOR_CASCADE_FILE = 'haarcascade_profileface.xml'
```

Imaging Section

ALLOWED_SOURCES

This configuration defines the source of the images that thumbor will load. This is only used in the HttpLoader (check the LOADER configuration above).

Plain string entries are matched literally against the request hostname. Dots are not wildcards.

```
ALLOWED_SOURCES = ['s.glbimg.com']
```

Regex-like string entries are accepted only for backward compatibility and will log a warning. For wildcard or regular expression entries, use compiled regular expression objects. Compiled patterns are matched against the full normalized image URL.

```
import re

ALLOWED_SOURCES = [
    re.compile(r'https?://[^/]+\.globo\.com/.*'),
    re.compile(r'https?://[^/]+\.glbimg\.com/.*'),
]
```

This is to get any images that are in `*.globo.com` or `*.glbimg.com` and fail with any other domains.

ACCESS_CONTROL_ALLOW_ORIGIN_HEADER

This allows to send the ACCESS_CONTROL_ALLOW_ORIGIN header. For example, if you want to tell the browser to allow code from any origin to access your thumbor resources:

```
ACCESS_CONTROL_ALLOW_ORIGIN_HEADER = '*'
```

If you want restrict access to a certain resource:

```
ACCESS_CONTROL_ALLOW_ORIGIN_HEADER = 'https://www.example.com'
```

Not set by default.

MAX_WIDTH and MAX_HEIGHT

These define the box that the resulting image for thumbor must fit-in. This means that no image that thumbor generates will have a width larger than MAX_WIDTH or height larger than MAX_HEIGHT. It defaults to 0, which means there is not limit. If the original image is larger than MAX_WIDTH x MAX_HEIGHT, it is proportionally resized to MAX_WIDTH x MAX_HEIGHT.

```
MAX_WIDTH = 1200
MAX_HEIGHT = 800
```

MIN_WIDTH and MIN_HEIGHT

These define the box that the resulting image for thumbor must fit-in. This means that no image that thumbor generates will have a width smaller than MIN_WIDTH or height smaller than MIN_HEIGHT. It defaults to 1. If the original image is smaller than MIN_WIDTH x MIN_HEIGHT, it is proportionally resized to MIN_WIDTH x MIN_HEIGHT.

```
MIN_WIDTH = 1
MIN_HEIGHT = 1
```

ALLOWED_SIZES

The sizes that image URLs may request, as `WIDTHxHEIGHT` strings. When the list is not empty, thumbor answers any other size with `400 Bad Request` before it loads the image. It defaults to an empty list, which allows every size.

Each entry must match the size in the URL exactly. A dimension that the URL omits or sets to `0` (proportional) counts as `0`, and `orig` counts as `orig`, so `/unsafe/400x/` requests `400x0`. A URL without a size requests `0x0`: add `"0x0"` to keep serving images without resizing them. Flips and `fit-in` do not change the comparison, so `-400x200` and `fit-in/400x200` both request `400x200`. thumbor refuses to start when an entry is not in this format.

```
ALLOWED_SIZES = ["400x200", "550x400", "1200x0"]
```

This option only restricts the size in the URL. The image that thumbor returns can still be smaller (with `fit-in`, for example), and unsigned URLs can still vary crop, filters and other options to create new images. To stop URL tampering, sign your URLs as described in *Security*.

QUALITY

This option defines the quality that JPEG images will be generated with. It defaults to 80.

```
QUALITY = 90
```

MAX_AGE

This option defines the number of seconds that images should remain in the browser's cache. It relates directly with the Expires and Cache-Control headers.

```
MAX_AGE = 24 * 60 * 60  # A day of caching
```

MAX_AGE_TEMP_IMAGE

When an image has some error in its detection or it has deferred queueing, it's convenient to set a much lower expiration time for the image cache. This way the browser will request the proper image faster.

This option defines the number of seconds that images in this scenario should remain in the browser's cache. It relates directly with the Expires and Cache-Control headers.

```
MAX_AGE_TEMP_IMAGE = 60  # A minute of caching
```

RESPECT_ORIENTATION

If this option is set to True, thumbor will reorient the image according to it's EXIF Orientation tag (if one can be found). This options defaults to False.

The operations performed in the image are as follow (considering the value of the Orientation EXIF tag):

1. Nothing
2. Flips the image horizontally
3. Rotates the image 180 degrees
4. Flips the image vertically
5. Flips the image vertically and rotates 270 degrees
6. Rotates the image 270 degrees
7. Flips the image horizontally and rotates 270 degrees
8. Rotates the image 90 degrees

```
RESPECT_ORIENTATION = False
```

ALLOW_ANIMATED_GIFS

This option indicates whether animated gifs should be supported.

```
ALLOW_ANIMATED_GIFS = True
```

ALLOW_ANIMATED_WEBP

Preserves animated WebP inputs when using the Pillow engine. Defaults to `True`. Frames, frame durations, loop count and transparency are retained when the output is WebP, including after resizing and filtering.

Set this option to `False` to process only the first frame, as in earlier versions. This setting is independent of `ALLOW_ANIMATED_GIFS` and `USE_GIFSICLE_ENGINE`. Static WebP inputs keep their existing behavior. An explicit `format(...)` conversion to another output format uses the first transformed frame. As with animated GIFs, the `AUTO_*` format options do not convert an animated WebP, so it is served as WebP whatever the request's `Accept` header. Animated PNG and AVIF inputs are not enabled by this option.

Every frame is decoded in full, so an animation whose frames hold more pixels in total than `MAX_PIXELS` allows for a single image is processed as its first frame.

```
ALLOW_ANIMATED_WEBP = True
```

USE_GIFSICLE_ENGINE

This option indicates whether gifsicle should be used for all gif images, instead of the actual imaging engine. This defaults to False.

When using gifsicle thumbor will generate proper animated gifs, as well as static gifs with the smallest possible size.

```
USE_GIFSICLE_ENGINE = True
```

WARNING: When using gifsicle engine, filters will be skipped, except for `cover()` filter. thumbor will not do smart cropping as well.

AUTO_*

These configurations indicates that thumbor will try to automatically convert the image format to a lighter image format, according to this compression order: `WEBP, AVIF, JPG, HEIF, PNG` — from highest (`WEBP`) to lowest (`PNG`) priority.

AUTO_WEBP

This option indicates whether thumbor should send WebP images automatically if the request comes with an "Accept" header that specifies that the browser supports "image/webp".

```
AUTO_WEBP = True
```

AUTO_AVIF

This option indicates whether thumbor should send Avif images automatically if the request comes with an "Accept" header that specifies that the browser supports "image/avif" and pillow-avif-plugin is enabled.

```
AUTO_AVIF = True
```

AUTO_PNG_TO_JPG

This option indicates whether thumbor should transform PNG images automatically to JPEG. If the image is a PNG without transparency and the numpy dependency is installed, thumbor will transform from png to jpeg. In the most of cases the image size will decrease.

WARNING: Depending on case, this is not a good deal. This transformation maybe causes distortions or the size of image can increase. Images with texts, for example, the result image maybe will be distorted. Dark images, for example,

the size of result image maybe will be bigger. You have to evaluate the majority of your use cases to take a decision about the usage of this conf.

This conversion remains independent of `AUTO_IMAGE_FORMAT_PREFERENCE`. When a preference list is configured and none of its entries can be used, thumbor still applies `AUTO_PNG_TO_JPG` as a fallback. The `autojpg()` filter enables or disables this fallback for an individual request; it does not enable or disable a `jpg` entry in the preference list.

```
AUTO_PNG_TO_JPG = True
```

AUTO_JPG

This option indicates whether thumbor should send JPG images automatically if the request comes with an "Accept" header that specifies that the browser supports `*/*`, `image/*`, `image/jpg` or `image/jpeg`.

```
AUTO_JPG = True
```

AUTO_PNG

This option indicates whether thumbor should send PNG images automatically if the request comes with an "Accept" header that specifies that the browser supports `image/png` or `image/*`.

```
AUTO_PNG = True
```

AUTO_HEIF

This option indicates whether thumbor should send Heif images automatically if the request comes with an "Accept" header that specifies that the browser supports "image/heif" and pillow-heif is enabled.

```
AUTO_HEIF = True
```

AUTO_IMAGE_FORMAT_PREFERENCE

This option defines the order in which thumbor attempts automatic output formats. The valid tokens are `webp`, `avif`, `jpg`, `heif` and `png`. Values are stripped of surrounding whitespace and converted to lowercase. Duplicate values after the first occurrence, non-string values and unknown tokens are ignored.

For each entry, thumbor checks the request's `Accept` header and the current engine's capabilities, then selects the first eligible format. AVIF and HEIF therefore require their respective engine support, JPEG is not selected for an image with transparency, and preference entries are not selected for multi-image input. Media types are matched case-insensitively, and an entry with an explicit quality value of `q=0` is not eligible. `image/*` applies only to PNG and JPEG, because browsers send it without being able to decode every format: WebP, AVIF and HEIF are selected only when the header lists `image/webp`, `image/avif` or `image/heif`. The less specific `*/*` retains the legacy behavior of enabling JPEG only. If no entry is eligible, thumbor preserves the engine's output format, except for the independent `AUTO_PNG_TO_JPG` fallback described above.

A preference list containing at least one valid entry overrides `AUTO_WEBP`, `AUTO_AVIF`, `AUTO_JPG`, `AUTO_PNG` and `AUTO_HEIF`. If any entries are invalid, thumbor logs one warning listing the ignored values. If every entry is invalid, thumbor treats the preference as empty. An empty preference uses the individual settings in their legacy order: `webp`, `avif`, `jpg`, `heif`, then `png`.

```
# Prioritize AVIF over WebP.
AUTO_IMAGE_FORMAT_PREFERENCE = ["avif", "webp", "jpg", "png", "heif"]
```

(continues on next page)

(continued from previous page)

```
# Or prioritize JPEG for workloads where that is preferable.
AUTO_IMAGE_FORMAT_PREFERENCE = ["jpg", "webp", "avif", "png", "heif"]
```

The result-storage cache key must vary with the active formats accepted by the request. The built-in file result storage does this automatically and uses an isolated namespace whenever a preference, `AUTO_AVIF`, `AUTO_JPG`, `AUTO_HEIF`, `AUTO_PNG` or `AUTO_PNG_TO_JPG` is active. It migrates legacy entries only for configurations where the old `default` or WebP namespace is unambiguous. Custom result storages must use `thumbor.auto_image_format.get_auto_image_format_cache_key` as described in *Custom Result Storages*. When enabling or reordering this setting, do not migrate a legacy cached object into a new format namespace unless its content type is verified; a cache purge is the safest upgrade path for storages that cannot isolate the old entries.

This namespace change also applies on upgrade when the preference remains empty but `AUTO_AVIF`, `AUTO_JPG`, `AUTO_HEIF`, `AUTO_PNG` or `AUTO_PNG_TO_JPG` is already enabled. Existing generated images in the legacy `default` namespace are deliberately treated as cache misses and regenerated. Plan for a temporarily cold cache, pre-warm it or use a gradual rollout if the additional origin and processing load would be significant. Legacy files are not removed automatically, so account for their disk usage until they are cleaned up separately. An `AUTO_WEBP`-only configuration keeps its existing cache namespace.

The default is an empty list, which retains the individual `AUTO_*` settings.

```
# Use individual AUTO_* settings (default behavior).
AUTO_IMAGE_FORMAT_PREFERENCE = []
```

Queueing - Redis Single Node

REDIS_QUEUE_MODE

Redis operation mode 'single_node' or 'sentinel'

```
REDIS_QUEUE_MODE = 'single_node'
```

REDIS_QUEUE_SERVER_HOST

Server host for the queued redis detector.

```
REDIS_QUEUE_SERVER_HOST = 'localhost'
```

REDIS_QUEUE_SERVER_PORT

Server port for the queued redis detector.

```
REDIS_QUEUE_SERVER_PORT = 6379
```

REDIS_QUEUE_SERVER_DB

Server database index for the queued redis detector

```
REDIS_QUEUE_SERVER_DB = 0
```

REDIS_QUEUE_SERVER_PASSWORD

Server password for the queued redis detector

```
REDIS_QUEUE_SERVER_PASSWORD = None
```

Queueing - Redis Sentinel

REDIS_QUEUE_MODE

Redis operation mode 'single_node' or 'sentinel'

```
REDIS_QUEUE_MODE = 'sentinel'
```

REDIS_QUEUE_SENTINEL_INSTANCES

Sentinel server instances for the queued redis detector.

```
REDIS_QUEUE_SENTINEL_INSTANCES = 'localhost:23679,localhost:23680'
```

REDIS_QUEUE_SENTINEL_PASSWORD

Sentinel server password for the queued redis detector.

```
REDIS_QUEUE_SENTINEL_PASSWORD = None
```

REDIS_QUEUE_SENTINEL_MASTER_INSTANCE

Sentinel server master instance for the queued redis detector.

```
REDIS_QUEUE_SENTINEL_MASTER_INSTANCE = 'masterinstance'
```

REDIS_QUEUE_SENTINEL_MASTER_PASSWORD

Sentinel server master password for the queued redis detector.

```
REDIS_QUEUE_SENTINEL_MASTER_PASSWORD = None
```

REDIS_QUEUE_SENTINEL_MASTER_DB

Sentinel server master database index for the queued redis detector.

```
REDIS_QUEUE_SENTINEL_MASTER_DB = 0
```

REDIS_QUEUE_SENTINEL_SOCKET_TIMEOUT

Sentinel server socket timeout for the queued redis detector.

```
REDIS_QUEUE_SENTINEL_SOCKET_TIMEOUT = 10.0
```

Queueing - Amazon SQS

This queue will be removed in an upcoming release in favor of the open source AWS plug-ins for thumbor.

SQS_QUEUE_KEY_ID

Amazon AWS key id.

```
SQS_QUEUE_KEY_ID = None
```

SQS_QUEUE_KEY_SECRET

Amazon AWS key secret.

```
SQS_QUEUE_KEY_SECRET = None
```

SQS_QUEUE_REGION

Amazon AWS SQS region.

```
SQS_QUEUE_REGION = 'us-east-1'
```

Security Section

SECURITY_KEY

This option specifies the security key that thumbor uses to sign secure URLs.

```
SECURITY_KEY = "replace-this-with-a-secret-key"
```

Do not use the example value in production.

ALLOW_UNSAFE_URL

This option specifies that the /unsafe url should be available in this thumbor instance. It is boolean (True or False).

> **Warning**
>
> It is **STRONGLY** recommended that you turn off this flag in production environments, as this can lead to DDoS attacks against thumbor.

```
ALLOW_UNSAFE_URL = False
```

Loader Options Section

FILE_LOADER_ROOT_PATH

In case you are using thumbor's built-in file loader, this is the option that allows you to specify where to find the images.

```
FILE_LOADER_ROOT_PATH = "/home/thumbor/images"
```

HTTP_LOADER_DEFAULT_USER_AGENT

This option allows users to specify the default User-Agent that thumbor sends when requesting images with the HTTP Loader. Its default is generated from the installed thumbor version, in the form `Thumbor/<version>`.

```
HTTP_LOADER_DEFAULT_USER_AGENT = "my-image-service/1.0"
```

HTTP_LOADER_FORWARD_USER_AGENT

This option tells thumbor to forward the request user agent when requesting images using the HTTP Loader. Defaults to False.

```
HTTP_LOADER_FORWARD_USER_AGENT = False
```

Storage Options Section

STORAGE_EXPIRATION_SECONDS

This options specifies the default expiration time in seconds for the storage.

```
STORAGE_EXPIRATION_SECONDS = 60  # 1 minute
```

With the file storage, `None` disables the expiration and `0` makes every file count as expired. The same expiration applies to uploaded images when `UPLOAD_PHOTO_STORAGE` is `thumbor.storages.file_storage` (directly or through `thumbor.storages.mixed_storage`), since they are kept in `FILE_STORAGE_ROOT_PATH` too. Expired files stay on disk until they are replaced or removed; see *Cleaning expired files*.

STORES_CRYPTO_KEY_FOR_EACH_IMAGE

This option specifies whether thumbor should store the key for each image (thus allowing the image to be found even if the security key changes). This is a boolean flag (True or False).

> **Warning**
>
> If this flag is set to False, it essentially means that whenever you change the security key, for whatever reason, you just invalidated every single image that's been generated before.
>
> That may be ok if you have another service fetching stored images instead of allowing thumbor to do it (as many of thumbor users do).

```
STORES_CRYPTO_KEY_FOR_EACH_IMAGE = True
```

File Storage Section

FILE_STORAGE_ROOT_PATH

In case you are using thumbor's built-in file storage, this is the option that allows you to specify where to save the images.

```
FILE_STORAGE_ROOT_PATH = '/home/thumbor/storage'
```

Result Storage Section

RESULT_STORAGE_EXPIRATION_SECONDS

Expiration in seconds of generated images in the result storage.

```
RESULT_STORAGE_EXPIRATION_SECONDS = 0
```

RESULT_STORAGE_FILE_STORAGE_ROOT_PATH

Path where the Result storage will store generated images.

```
RESULT_STORAGE_FILE_STORAGE_ROOT_PATH = '/tmp/thumbor/result_storage'
```

RESULT_STORAGE_STORES_UNSAFE

Indicates whether unsafe requests should also be stored in the Result Storage.

```
RESULT_STORAGE_STORES_UNSAFE = False
```

Healthcheck

HEALTHCHECK_ROUTE

The URL path to a healthcheck. This will return a 200 and the text 'WORKING'.

```
HEALTHCHECK_ROUTE = '/status'
```

Will put the healthcheck response on `http://host:port/status`

The default route is `/healthcheck`.

Logging

THUMBOR_LOG_FORMAT

This option specifies the format to be used by logging messages sent from thumbor.

```
THUMBOR_LOG_FORMAT = '%(asctime)s %(name)s:%(levelname)s %(message)s'
```

THUMBOR_LOG_DATE_FORMAT

This option specifies the date format to be used by logging messages sent from thumbor.

```
THUMBOR_LOG_DATE_FORMAT = '%Y-%m-%d %H:%M:%S'
```

Error Handling

USE_CUSTOM_ERROR_HANDLING

This configuration indicates whether thumbor should use a custom error handler.

```
USE_CUSTOM_ERROR_HANDLING = False
```

ERROR_HANDLER_MODULE

Error reporting module. Needs to contain a class called ErrorHandler with a handle_error(context, handler, exception) method.

```
ERROR_HANDLER_MODULE = 'thumbor.error_handlers.sentry'
```

Error Handling - Sentry

SENTRY_DSN_URL

Sentry thumbor project DSN, for example: `http://user:password@localhost:9000/2`.

```
SENTRY_DSN_URL = ''
```

SENTRY_ENVIRONMENT

Sentry thumbor environment.

```
SENTRY_ENVIRONMENT = 'staging'
```

Upload

UPLOAD_MAX_SIZE

Max size in bytes for images uploaded to thumbor.

```
UPLOAD_MAX_SIZE = 0
```

UPLOAD_ENABLED

Indicates whether thumbor should enable File uploads.

```
UPLOAD_ENABLED = False
```

UPLOAD_PHOTO_STORAGE

The type of storage to store uploaded images with.

```
UPLOAD_PHOTO_STORAGE = 'thumbor.storages.file_storage'
```

UPLOAD_DELETE_ALLOWED

Indicates whether image deletion should be allowed.

```
UPLOAD_DELETE_ALLOWED = False
```

UPLOAD_PUT_ALLOWED

Indicates whether image overwrite should be allowed.

```
UPLOAD_PUT_ALLOWED = False
```

UPLOAD_DEFAULT_FILENAME

Default filename for image uploaded.

```
UPLOAD_DEFAULT_FILENAME = 'image'
```

GC_INTERVAL

`GC_INTERVAL` is retained in the configuration registry for compatibility, but the current Thumbor server does not consume it. Setting it has no effect. Use runtime and container memory controls instead of relying on this option.

Generate a configuration template

Generate the configuration template for the installed thumbor version with:

```
thumbor-config > ./thumbor.conf
```

The generated file is the authoritative list of built-in settings, defaults, aliases and descriptions for that installed version. Keep custom Python code, plugin settings and local overrides in your own `thumbor.conf`; regenerating the template overwrites the destination selected by shell redirection.

2.6.2 Cleaning expired files

The file storage and the file result storage never delete expired files. An expired file is ignored, and it is only replaced when the same image is requested again, so files that are not requested anymore stay on disk forever.

`thumbor-cleanup` deletes them. Run it periodically from the same machine, or the same volumes, as thumbor:

```
# Report what would be deleted
thumbor-cleanup -c /etc/thumbor.conf --result-storage

# Delete it
thumbor-cleanup -c /etc/thumbor.conf --result-storage --delete
```

Without `--delete` it only reports the files it would delete. It reads the same configuration file as the server; when `-c` is omitted it looks for `thumbor.conf` the same way the server does, and `--use-environment` enables the same environment overrides. Those only work for string settings such as the root paths, so keep the expirations in the configuration file, or read the environment inside it as described in *Converting environment values*.

Options

- `-c`, `--conf`: the thumbor configuration file.
- `--use-environment`: read string settings from environment variables named after them, like the server does. `--use-environment=True` is accepted too.
- `--storage`: clean `FILE_STORAGE_ROOT_PATH` using `STORAGE_EXPIRATION_SECONDS`.
- `--result-storage`: clean `RESULT_STORAGE_FILE_STORAGE_ROOT_PATH` using `RESULT_STORAGE_EXPIRATION_SECONDS`.
- `--delete`: delete the files instead of only reporting them.
- `--include-uploads`: allow `--storage` when uploads are stored in the file storage. See below.
- `--older-than SECONDS`: use this age instead of the configured expiration. `--grace` is still added to it, so pass `--grace 0` to use the age as is.
- `--grace SECONDS`: extra age added to the expiration before a file is deleted. Defaults to 3600.

- `--temp-older-than SECONDS`: age after which leftover temporary files are deleted. Defaults to 3600.
- `-v`, `--verbose`: print every file that is deleted or would be deleted.

At least one of `--storage` and `--result-storage` is required. The command exits with 0 on success, 1 when a file or directory could not be read or deleted, and 2 for invalid arguments or a configuration it refuses to clean.

What is deleted

A file is deleted when it is older than the expiration plus the grace period. thumbor already treats such a file as expired and does not read it, so the grace period keeps the command from removing a file that a running server has just checked and is about to read. The file storage and the file result storage also treat a file that disappears while they read it as missing, so a request that races the command (for example with an `--older-than` shorter than the configured expiration) gets a cache miss instead of an error.

When the expiration is `None`, zero or negative, only leftover temporary files are deleted from that root unless `--older-than` is given. Note that thumbor itself treats a `STORAGE_EXPIRATION_SECONDS` of zero as "every file is expired", while `RESULT_STORAGE_EXPIRATION_SECONDS` of zero means "never expires"; the cleanup command never reads a zero as a reason to delete everything.

In the storage root, the command only considers the files the file storage writes: images at `<2 hex digits>/<38 hex digits>`, their security key (`.txt`) and detector (`.detectors.txt`) files, and temporary files. Anything else under that root is left alone. The detector files of an image are deleted together with it, and those whose image no longer exists are deleted once they are old enough on their own. Security key files are never deleted while `STORES_CRYPTO_KEY_FOR_EACH_IMAGE` is on: thumbor does not expire them, and they are what keeps a URL signed before `SECURITY_KEY` changed valid, even after the image itself is gone. With that option off thumbor never reads them, so they are deleted like the detector files. The blacklist file is never deleted, since it is configuration, not cache.

In the result root every regular file is considered, whatever its name, because the legacy layout names files after the request. In both roots, temporary files left behind by an interrupted write are deleted after `--temp-older-than` seconds, even when the expiration is disabled.

The command does not follow symbolic links found inside a root, only deletes regular files, never leaves the configured root and keeps every directory. A root that is itself a symbolic link is cleaned at its target, which is where thumbor reads and writes those files. When one root is inside the other, cleaning the outer one skips the inner one. Each file is checked again right before it is deleted, and one that was stored again since its directory was listed is kept. Files that disappear while the command runs are ignored.

Uploaded images

When `UPLOAD_ENABLED` is on and `UPLOAD_PHOTO_STORAGE` is `thumbor.storages.file_storage`, or `thumbor.storages.mixed_storage` with `MIXED_STORAGE_FILE_STORAGE` set to the file storage, uploaded images live in `FILE_STORAGE_ROOT_PATH` next to the cached original images and expire with the same `STORAGE_EXPIRATION_SECONDS`. They cannot be told apart on disk, so `--storage` refuses to run unless `--include-uploads` is given. With it, every uploaded image older than the expiration is deleted for good. Raise `STORAGE_EXPIRATION_SECONDS` or leave the storage root alone if uploads must be kept.

Scheduling

With cron, for example in `/etc/cron.d/thumbor-cleanup`:

```
17 3 * * * thumbor thumbor-cleanup -c /etc/thumbor.conf --result-storage --delete
```

With a systemd timer, `/etc/systemd/system/thumbor-cleanup.service`:

```
[Unit]
Description=Delete expired thumbor files
```

(continues on next page)

(continued from previous page)

```
[Service]
Type=oneshot
User=thumbor
ExecStart=/usr/local/bin/thumbor-cleanup -c /etc/thumbor.conf --result-storage --delete
```

and `/etc/systemd/system/thumbor-cleanup.timer`:

```
[Unit]
Description=Run thumbor-cleanup daily

[Timer]
OnCalendar=daily
RandomizedDelaySec=1h
Persistent=true

[Install]
WantedBy=timers.target
```

Then enable it with `systemctl enable --now thumbor-cleanup.timer`.

With a Kubernetes CronJob that mounts the same configuration and volume as the thumbor deployment:

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: thumbor-cleanup
spec:
  schedule: "17 3 * * *"
  concurrencyPolicy: Forbid
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: OnFailure
          containers:
            - name: thumbor-cleanup
              image: ghcr.io/thumbor/thumbor:latest
              command:
                - thumbor-cleanup
                - -c
                - /etc/thumbor.conf
                - --result-storage
                - --delete
              volumeMounts:
                - name: config
                  mountPath: /etc/thumbor.conf
                  subPath: thumbor.conf
                - name: result-storage
                  mountPath: /data/result_storage
          volumes:
            - name: config
              configMap:
```

(continues on next page)

(continued from previous page)

```
            name: thumbor-config
        - name: result-storage
          persistentVolumeClaim:
            claimName: thumbor-result-storage
```

Without thumbor-cleanup

The result storage holds nothing but generated images, so `find` is enough there. For a one day expiration plus one hour of grace (1500 minutes):

```
find /data/result_storage -type f -mmin +1500 -delete
```

Do not use this on the storage root: it also holds uploads, security key and detector files, and the blacklist.

2.6.3 Automated Error Handling

For many companies it make a lot of sense to have a centralized solution for handling errors in production, like sentry or squash.

Thumbor must support this type of error handling in order to better integrate itself to it's users environments.

Enabling Custom Error Handling

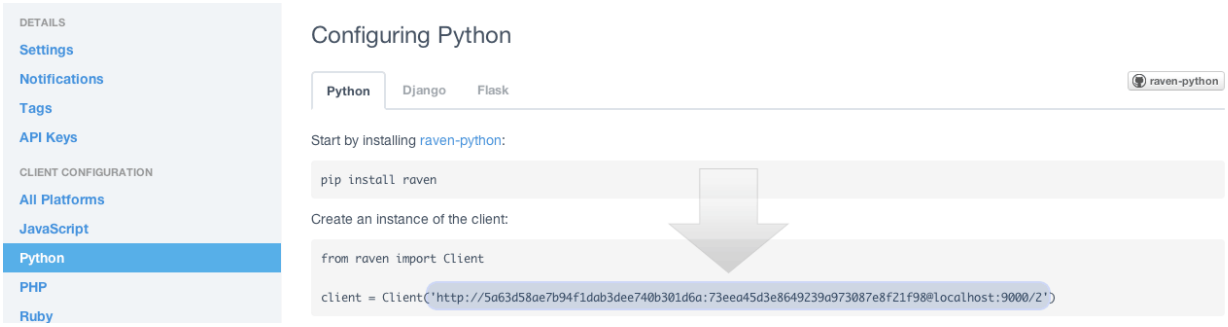
Enabling it is as simple as setting the configuration `USE_CUSTOM_ERROR_HANDLING` to `True`.

After that you need to set the custom error handler you want to use with the `ERROR_HANDLER_MODULE` configuration. Please note that this is the module full name, not the class full name.

Thumbor comes pre-packaged with sentry's custom error handler: `thumbor.error_handlers.sentry`. If you decide to use it, please read below on how to configure it.

Sentry - thumbor.error_handlers.sentry

If you choose to use sentry custom error handler, all you need to do is fill the `SENTRY_DSN_URL` configuration with sentry's DSN URL, which can be found in the admin page for your sentry project, like the one in the image below:



2.6.4 Hosting

Let's see how would you run thumbor in different environments.

Development Environment

For local development, put `thumbor.conf` in the current working directory, the current user's home directory, or `/etc/thumbor.conf`. If no file exists in those locations, thumbor uses the configuration bundled with the installed package. You can select another file with `-c` or `--conf`; see *Configuration* for the complete lookup order.

To verify if you have thumbor, just type:

```
thumbor --version
```

It should return the version you've installed. Starting thumbor is as easy as:

```
thumbor
```

For more options check the *Configuration* page.

Production Environment

Other than having the proper *Configuration* file for your environment, we have some recommendations on how to run thumbor in production.

Our first recommendation is to run more than one instance of it. You can specify different ports using thumbor easily. This will make sure that your service stays responsive even if one of the processes die.

We also recommend having some form of load balance that distributes the load between the aforementioned processes. We are using NGINX to do it, but there are more sophisticated load balance softwares around. thumbor supports health checking under the `/healthcheck` URI if you need to use it.

Other than that, you run it using the thumbor console app specifying the arguments, like this:

```
thumbor --port=8888 --conf="$HOME/mythumbor.conf"
```

We recommend using an application such as Supervisor (http://supervisord.org/index.html) to monitor your services. An example of a `supervisord.conf` file would be:

```
[supervisord]
logfile = /home/thumbor/logs/supervisord.log
logfile_maxbytes = 50MB
logfile_backups=10
loglevel = info
pidfile = /home/thumbor/supervisord.pid
user = thumbor

[program:thumbor]
command=thumbor --port=800%(process_num)s --conf=/etc/thumbor800%(process_num)s.conf
process_name=thumbor800%(process_num)s
numprocs=4
user=thumbor
directory=/home/thumbor/
autostart=true
autorestart=true
startretries=3
stopsignal=TERM
stdout_logfile=/home/thumbor/logs/thumbor800%(process_num)s.stdout.log
stdout_logfile_maxbytes=1MB
stdout_logfile_backups=10
stderr_logfile=/home/thumbor/logs/thumbor800%(process_num)s.stderr.log
```

(continues on next page)

(continued from previous page)

```
stderr_logfile_maxbytes=1MB
stderr_logfile_backups=10
```

This configuration file makes sure that supervisor starts 4 processes of thumbor on the 8000, 8001, 8002 and 8003 ports, each with a different configuration file (thumbor8000.conf, thumbor8001.conf, thumbor8002.conf, thumbor8003.conf all under /etc folder). The other settings are optional, but if you need help with supervisor's settings it has extensive documentation online (http://supervisord.org/introduction.html).

Thumbor in the Cloud

Running with Docker

Running thumbor with an official Docker image is very straightforward. An official image is built and hosted on GitHub Container Registry.

```
docker run -p 8888:8888 ghcr.io/thumbor/thumbor:latest
```

To verify thumbor is running, you can check the health endpoint:

```
curl http://localhost:8888/healthcheck
```

For more information about the official Docker image, visit the GitHub Container Registry.

Configuring the Docker Image

The official image uses `thumbor` as its entry point. Pass thumbor command-line arguments after the image name.

To use a configuration file, mount it at one of thumbor's normal lookup locations:

```
docker run -p 8888:8888 \
  -v /path/to/thumbor.conf:/etc/thumbor.conf:ro \
  ghcr.io/thumbor/thumbor:latest
```

Alternatively, mount it at another location and select it explicitly:

```
docker run -p 8888:8888 \
  -v /path/to/thumbor.conf:/opt/thumbor/thumbor.conf:ro \
  ghcr.io/thumbor/thumbor:latest \
  --conf=/opt/thumbor/thumbor.conf
```

The official image does not translate `THUMBOR_*` environment variables into thumbor configuration. If you explicitly enable the core legacy environment mode, use exact, unprefixed configuration names and remember that their values remain strings; see *Configuration*.

Generate a template with `thumbor-config > thumbor.conf` on an installed copy of thumbor.

For more detailed configuration options, see the *Configuration* documentation.

Docker Compose configuration

For string settings, enable environment overrides in Compose with:

```
services:
  thumbor:
    image: ghcr.io/thumbor/thumbor:latest
```

(continues on next page)

(continued from previous page)

```
    ports:
      - "8888:8888"
    command: ["--use-environment=True"]
    environment:
      SECURITY_KEY: "replace-with-your-own-key"
```

The `command` list supplies arguments to the image's `thumbor` entrypoint. Quote environment values so Compose treats them as strings.

For settings that need other types, use the Python configuration example in *Converting environment values* and mount it as follows:

```
services:
  thumbor:
    image: ghcr.io/thumbor/thumbor:latest
    ports:
      - "8888:8888"
    command: ["-c", "/etc/thumbor.conf"]
    environment:
      SECURITY_KEY: "replace-with-your-own-key"
      UPLOAD_ENABLED: "true"
      UPLOAD_DELETE_ALLOWED: "false"
      UPLOAD_PUT_ALLOWED: "false"
      QUALITY: "85"
    volumes:
      - ./thumbor.conf:/etc/thumbor.conf:ro
```

Here the file reads `SECURITY_KEY` as a string and converts the other values to Python booleans and integers. Keep `--use-environment` out of this command to preserve those types.

Thumbor on OpenShift

> **Warning**
>
> This may be outdated since thumbor moved to python 3.

There's a project showing how to deploy a working version on OpenShift https://github.com/rafaelcaricio/thumbor-openshift-example

Thumbor behind CloudFront

> **Warning**
>
> This may be outdated since thumbor moved to python 3.

The awesome people at yipit are using thumbor behind the CloudFront CDN at Amazon.

The detailed information on how to do it can be seen at this blog post.

2.6.5 Logging

thumbor uses Python's built-in logging system. The `-l` or `--log-level` command-line option selects the level; see *Running thumbor server*.

Logging configuration

Logging settings live in `thumbor.conf` like other configuration keys.

THUMBOR_LOG_CONFIG

A Python dictionary accepted by `logging.config.dictConfig`. When this setting is a non-empty dictionary, thumbor applies it and does not call `logging.basicConfig`; `THUMBOR_LOG_FORMAT`, `THUMBOR_LOG_DATE_FORMAT` and the command-line log level therefore do not configure the handlers created by this dictionary. `None`, an empty string or an empty dictionary fall back to `logging.basicConfig`.

```
THUMBOR_LOG_CONFIG = {
    "version": 1,
    "disable_existing_loggers": False,
    # Add formatters, handlers, loggers and root here.
}
```

The default is `None`.

THUMBOR_LOG_FORMAT

Format used when `THUMBOR_LOG_CONFIG` is not set.

```
THUMBOR_LOG_FORMAT = '%(asctime)s %(name)s:%(levelname)s %(message)s'
```

THUMBOR_LOG_DATE_FORMAT

Date format used when `THUMBOR_LOG_CONFIG` is not set.

```
THUMBOR_LOG_DATE_FORMAT = '%Y-%m-%d %H:%M:%S'
```

2.6.6 Running thumbor server

Running thumbor server is as easy as typing "thumbor" (considering you went through the proper *Installing* procedures).

The Server application takes some parameters that will help you tailor the thumbor Server to your needs. If you want to find out what the thumbor Server arguments are, just type:

```
thumbor --help
```

-i or –ip

The address that Tornado will listen for incoming request. It defaults to *0.0.0.0* (listening on localhost and current IP).

-p or –port

The port that Tornado will listen for incoming request. It defaults to *8888*.

-c or –conf

Path to the configuration file for this server. The path may be absolute or relative to the current working directory. Shell shortcuts such as ~ are not expanded by thumbor; use an expanded path such as `--conf="$HOME/mythumbor.conf"`.

If this option is omitted, thumbor searches for `thumbor.conf` as described in *Configuration*.

-k or –keyfile

The path for the file containing the security key to be used for this server. See *Loading the key from a file* for the paths thumbor accepts.

-l or –log-level

The log level to be used. Possible values are: *debug*, *info*, *warning*, *error*, *critical* or *notset*. More on that at http://docs.python.org/library/logging.html. It defaults to *warning*.

–use-environment

Enables legacy environment overrides. This option currently requires a value, for example `--use-environment=true`. Variable names are unprefixed and must exactly match configuration keys. Values remain strings. For names, typing and precedence, see *Configuration*.

–processes

Number of processes to run. By default equals 1 and means no forks created. Set to 0 to detect the number of cores available on this machine. Set > 1 to start that specified number of processes.

-a or –app

Allows the user to specify the application class to be used. This is a very advanced usage of thumbor. This argument is specified like: "namespace1.namespace2.class_name" as in "myproj.thumbor_support.MyProjThumborApp".

Signing thumbor urls

To help users create signed URLs (mostly for debugging purposes, since we recommend using the *Libraries*), thumbor comes with an application called thumbor-url.

In order to use it, type `thumbor-url -h` and it will present all options available.

Cleaning expired files

The file storage and the file result storage keep expired files on disk. thumbor comes with `thumbor-cleanup` to delete them; see *Cleaning expired files*.

2.6.7 Image Metadata

Thumbor uses piexif to read and write image metadata.

The image metadata is available in `engine.metadata`.

Reading and writing Metadata

Let's retrieve a list of all the available EXIF tags available in the image:

```
>>> engine.metadata
{
  '0th': {
      271: b'Canon',
      272: b'Canon EOS 5D Mark III',
      282: (300, 1),
      283: (300, 1),
      296: 2,
      305: b'Adobe Photoshop Lightroom 4.4 (Macintosh)',
      306: b'2016:06:24 14:45:44',
      34665: 216
  },
  'Exif': {
      33434: (1, 100),
      33437: (56, 10),
      34850: 1,
      34855: 640,
      34864: 2,
      34866: 640,
      36864: b'0230',
      36867: b'2016:06:23 13:18:05',
      36868: b'2016:06:23 13:18:05',
      37377: (6643856, 1000000),
      37378: (4970854, 1000000),
      37380: (0, 1),
      37381: (3, 1),
      37383: 5,
      37385: 16,
      37386: (123, 1),
      37521: b'91',
      37522: b'91',
      41486: (5242880, 32768),
      41487: (5242880, 32768),
      41488: 4,
      41985: 0,
      41986: 1,
      41987: 1,
      41990: 0,
      42033: b'042024004240',
      42034: ((70, 1), (200, 1), (0, 0), (0, 0)),
      42036: b'EF70-200mm f/2.8L IS II USM',
      42037: b'0000c139be'},
  'GPS': {},
  'Interop': {},
  '1st': {},
  'thumbnail': None
}
```

The reference to the values can be found in Exif values.

```
>>> tag = metadata["Exif"][piexif.ExifIFD.DateTimeOriginal]
"2016:06:23 13:18:05"
```

piexif API reference

See the piexif documentation for its full API.

2.6.8 Security

thumbor's team is very concerned about security and vulnerabilities of the service. Even though the team strives to cover most scenarios, if you find any flaws or vulnerabilities, please contact the team or create an issue.

URL Tampering

Consider the following URL for an image: `http://some.server.com/unsafe/300x300/smart/path/to/image.jpg`.

Now let's say that some malicious user wants to overload your service. He can easily ask for other sizes in loops or worse, like:

```
http://some.server.com/unsafe/300x301/smart/path/to/image.jpg
http://some.server.com/unsafe/300x302/smart/path/to/image.jpg
http://some.server.com/unsafe/300x303/smart/path/to/image.jpg
...
http://some.server.com/unsafe/300x9999/smart/path/to/image.jpg
...
http://some.server.com/unsafe/9999x9999/smart/path/to/image.jpg
```

And that's not even counting varying the available options.

Other than that, the user can ask for images that do not exist, thus forcing us to perform useless http GET operations or filesystem operations.

We classified both scenarios above as **URL Tampering**.

Stopping Tampering

In order to prevent users from tampering with the URL, thumbor provides a configuration called `SECURITY_KEY`. This is the key used to generate a hash-based message authentication code.

The process is very straightforward. The web server that has the page using thumbor's image generates an authentication code for the options and image url, using the **SECURITY_KEY**.

When end-users access the page and thus load the image, thumbor generates an authentication code for the same options and image url, using the same **SECURITY_KEY**. If both authentication codes match, thumbor processes it.

The secure endpoint looks like this: `/<authentication code with 28 characters>/300x200/smart/path/to/image.jpg`.

Loading the key from a file

The key can also come from a file, through the `-k/--keyfile` server option. When set, the key file takes precedence over `SECURITY_KEY`. Before reading the file, thumbor checks its path:

- An absolute path works from any working directory, but the file, symlinks followed, must stay under the path's parent directory: `/etc/thumbor/thumbor.key` may link to `/etc/thumbor/keys/current` but not to `/root/thumbor.key`.
- A relative path, symlinks followed, must stay under the directory thumbor was started from.

A path that breaks either rule stops thumbor at startup.

HMAC method

We intend to supply toolkits in many languages that automate the signing process, but we might need help from the community in this direction.

thumbor uses standard HMAC with SHA1 signing.

Let's use as an example the url `http://some.server.com/unsafe/300x200/smart/path/to/image.jpg`.

In order to convert that to a "safe" url, we must sign the part `300x200/smart/path/to/image.jpg`:

1. Generate a `signature` of that part using HMAC-SHA1 with the **SECURITY_KEY**.
2. Encode the `signature` as base64. thumbor uses `urlsafe_b64encode` method of the native python's base64 module. This method replaces some characters in the base64 string so it becomes more url friendly.
3. Append the `encoded_signature` to the beginning of the URL, like: `/12345678901234567890123456789012/300x200/smart/path/to/image.jpg`.

That last part gives you the new URL: `http://thumbor-server/12345678901234567890123456789012/300x200/smart/path/to/image.jpg`. Notice that the url includes the options part `300x200/smart`. That's required for thumbor to generate an authentication code to match the one that signs the image (`12345678901234567890123456789012`).

The code included in this documentation is illustrational and should not be used for any purposes.

The description of the base64 method is: reference

```
base64.urlsafe_b64encode(s)
Encode string s using a URL-safe alphabet, which substitutes
- instead of + and _ instead of / in the standard Base64 alphabet.
The result can still contain =.
```

Loading Images over HTTPS

The default http_loader loads images by default over http. To change the default to https, use the https_loader instead. To enforce https, use the strict_https_loader. Check the *Image loader* page for more details.

Libraries

There are implementations of url generators in various languages, take a look at the *Libraries* page to find information about them.

2.7 Upload

> **Warning**
>
> The upload module will be removed from thumbor's codebase soon and ported to an extension.

2.7.1 How to upload Images

Thumbor provides a `/image` REST end-point to upload your images and manage it.

This way you can send thumbor your original images by doing a simple post to its urls.

Configuration

The table below show all configuration parameters to manage image upload:

Configuration parameter	Default	Description
UPLOAD_ENABLED	False	Indicates whether thumbor should enable File uploads
UPLOAD_PUT_ALLOWED	False	Indicates whether image overwrite should be allowed
UPLOAD_DELETE_ALLOWED	False	Indicates whether image deletion should be allowed
UPLOAD_PHOTO_STORAGE	thumbor.storages.file_storage	The type of storage to store uploaded images with
UPLOAD_DEFAULT_FILENAME	image	Default filename for image uploaded
UPLOAD_MAX_SIZE	0	Max size in bytes for images uploaded to thumbor
MIN_WIDTH	1	Min width in pixels for images uploaded
MIN_HEIGHT	1	Min height in pixels for images uploaded

Here, `file_storage` means `thumbor.storages.file_storage`.

Thumbor comes with the `/image` REST end-point to upload disabled by default. In order to enable it, just set the `UPLOAD_ENABLED` configuration in your thumbor.conf file to `True`.

Thumbor will then use the storage specified in the `UPLOAD_PHOTO_STORAGE` configuration to save your images. You can use an existing storage (filesystem, redis, mongo, hbase...) or *create your own storage* if needed .

You can manage image putting and deletions simply set the configuration parameters `UPLOAD_PUT_ALLOWED` and `UPLOAD_DELETE_ALLOWED` to `True`. This parameters are set to `False` by default for security reasons.

Finally the upload constraints (max size, image width and height) will be controlled by `UPLOAD_MAX_SIZE`, `MIN_WIDTH` and `MIN_HEIGHT` parameters.

API Usage

The Thumbor `/image` REST end-point supports the commons HTTP methods :

- POST : to upload a new image
- GET : to display an image uploaded
- PUT : to replace an image uploaded by another preserving the URI
- DELETE : to remove an image uploaded from storage

By default, `PUT` and `DELETE` methods are disabled as explained above. This is done to tighten thumbor's security.

Posting

Posting is the only method allowed by default when you activate the upload module. It allows new images to be sent to Thumbor.

In order to upload a new image, you have two choices:

- send an HTTP **POST** to the `/image` end-point with the image as payload (REST style)
- send an HTTP **POST** to the `/image` end-point with a multi-part form with a file field called media (Form style).

In the REST style mode you may add an optional `Slug` header to define the image filename, which is useful for SEO reasons. As RFC 5023 defines it, the `Slug` value is percent-encoded UTF-8, so `photo%20name.jpg` names

the image `photo name.jpg`. Not specifying a `Slug` causes the server to use the default filename for the image (`UPLOAD_DEFAULT_FILENAME` parameter) .

The HTTP response will return a `Location` header pointing on the uploaded image, with the filename percent-encoded. The URI presented in `Location` header may be used to update or delete the image uploaded (see below).

For examples, see *Upload an image via the REST API* or *Upload an image via a form*.

HTTP status code

The status code returned will be :

- 201 Created (success)
- 415 Unsupported Media Type (image type is not allowed)
- 412 Precondition Failed (image is too small or the file is not an image)

Putting

Putting is a little more dangerous if you don't have strict control of who can access the `/image` end-point. This is because whatever is sent using this method gets saved to storage, overwriting the previous entry.

In order to replace an existing image, all you have to do is send an HTTP **PUT** request to the `/image` end-point with the new image content as payload. The new image will replace the original image preserving the URI.

As for the `POST` method you may add an optional `Slug` header to define the image filename.

The HTTP response will return a `Location` header pointing on the modified image. The URI presents in `Location` header may be used to update again the image or delete it.

For an example, see *Modifying an image*.

HTTP status code

The status code returned will be :

- 204 No Content (success)
- 405 Method Not Allowed (if thumbor's configuration disallows putting images)
- 415 Unsupported Media Type (image type is not allowed)
- 412 Precondition Failed (image is too small or file is not an image)

Deleting

Deleting can be very dangerous, thus is disabled by default.

If you do enable it, in order to delete an image, all you have to do is send an HTTP **DELETE** request to the `/image` end-point.

For an example, see *Deleting an image*.

HTTP status code

- 204 No Content (success)
- 404 Not Found (image doesn't exists)
- 405 Method Not Allowed (if thumbor's configuration disallows deleting images)

Example

Assuming the thumbor server is located at : `http://thumbor-server`

Upload an image via the REST API

When using the `/image` REST end-point to upload your image via the REST API :

```
curl -i -H "Content-Type: image/jpeg" -H "Slug: photo.jpg" \
    -XPOST http://thumbor-server/image \
    --data-binary "@tests/fixtures/images/20x20.jpg"
```

the HTTP **POST** request was send to the server :

```
POST /image
Content-Type: image/jpeg
Content-Length: 822
Slug : photo.jpg
```

and the Thumbor server should return:

```
HTTP/1.1 201 Created
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Location: /image/05b2eda857314e559630c6f3334d818d/photo.jpg
Server: TornadoServer/2.1.1
```

The image is created at `http://thumbor-server/image/05b2eda857314e559630c6f3334d818d/photo.jpg`. It can be retrieved, modified or deleted via this URI.

The optional `Slug` HTTP header specifies the filename to use for the image uploaded.

Upload an image via a form

When using the `/image` REST end-point to upload your images via a form, the user is free to choose the filename of the image via the `filename` field :

```
curl -i -XPOST http://thumbor-server/image \
    -F "media=@tests/fixtures/images/20x20.jpg;type=image/jpeg;filename=croco.jpg"
```

the HTTP **POST** request was send to the server :

```
POST /image
Content-Type: multipart/form-data; boundary=----------------------------11df125d8b12
Content-Length: 822
```

and the Thumbor server should return:

```
HTTP/1.1 201 Created
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Location: /image/05b2eda857314e559630c6f3334d818d/croco.jpg
```

The image is created at `http://thumbor-server/image/05b2eda857314e559630c6f3334d818d/croco.jpg`. It can be retrieve, modify or delete via this URI using the REST API.

Modifying an image

To replace the previously uploaded image by another we use:

```
curl -i -H "Content-Type: image/jpeg" -H "Slug: modified_image.jpg" \
    -XPUT http://thumbor-server/image/05b2eda857314e559630c6f3334d818d/photo.jpg \
    --data-binary "@tests/fixtures/images/20x20.jpg"
```

the HTTP **PUT** request was send to the server :

```
PUT /image/05b2eda857314e559630c6f3334d818d/photo.jpg
Content-Type: image/jpeg
Content-Length: 822
Slug : modified_image.jpg
```

and the Thumbor server should return:

```
HTTP/1.1 204 No Content
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Location: /image/05b2eda857314e559630c6f3334d818d/modified_image.jpg
Server: TornadoServer/2.1.1
```

Deleting an image

Finally to delete the uploaded image we use:

```
curl -i -XDELETE \
    http://thumbor-server/image/05b2eda857314e559630c6f3334d818d/modified_image.jpg
```

the HTTP **DELETE** request was send to the server :

```
DELETE /image/05b2eda857314e559630c6f3334d818d/modified_image.jpg
```

and the Thumbor server should return:

```
HTTP/1.1 204 No Content
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Server: TornadoServer/2.1.1
```

2.7.2 Posting, Putting and Deleting

thumbor original photo uploading end-point supports three different http verbs: put, post and delete.

By default, put and delete are disabled. This is done to tighten thumbor's security. If you wish to enable them, please refer to the *How to upload Images* page.

Posting

Posting is the only allowed by default method. It allows new images to be sent to thumbor. If the same image is sent again, thumbor will raise an exception.

This is done so users can't overwrite images with other images, possibly defacing your website.

In order to post a new image, all you have to do is send a multi-part form with a file field called media and action of `http://{thumbor-server}/image` and method of `POST`.

HTTP status code

- 201 Created (success)
- 409 Conflict (image already exists)
- 412 Precondition Failed (image is too small or the file is not an image)

Putting

Putting is a little more dangerous if you don't have strict control of who can access the `/image` route. This is because whatever is sent using this method gets saved to storage, overwriting the previous entry.

In order to put a new image, all you have to do is send a multi-part form with a file field called `media` and action of `http://{thumbor-server}/image` and method of `PUT`.

HTTP status code

- 201 Created (success)
- 405 Method Not Allowed (if thumbor's configuration disallows putting images)
- 412 Precondition Failed (image is too small or file is not an image)

Deleting

Deleting can be very dangerous, thus is disabled by default.

If you do enable it, in order to delete an image, all you have to do is send a request to `http://{thumbor-server}/image` with a method of `DELETE` and a field called `file_path` with the same path that was used when uploading the image.

HTTP status code

- 200 OK (success)
- 405 Method Not Allowed (if thumbor's configuration disallows deleting images)

2.7.3 Creating an Upload Storage

The upload API writes new and replacement images through the module configured by `UPLOAD_PHOTO_STORAGE`. Thumbor imports a class named `Storage` from that module and instantiates it with the current request context.

An upload storage should inherit from `thumbor.storages.BaseStorage` and implement an asynchronous `put` method:

```
from thumbor.storages import BaseStorage


class Storage(BaseStorage):
    async def put(self, path, file_bytes):
        """Store file_bytes under path."""
```

Configure the module name, without the `.Storage` class suffix:

```
UPLOAD_PHOTO_STORAGE = "my_package.upload_storage"
```

The module must be importable in the Python environment that runs Thumbor. `ORIGINAL_PHOTO_STORAGE` remains a compatibility alias, but new configurations should use `UPLOAD_PHOTO_STORAGE`.

The built-in `/image/<id>` retrieval and deletion handlers use the main `STORAGE`, not `UPLOAD_PHOTO_STORAGE`. To manage uploaded images through those routes, configure compatible backends for both settings. If the same module is used as the main `STORAGE`, it must also implement `get`, `exists`, `remove` and the metadata operations required by the enabled storage features. See the custom storage documentation for that complete interface.

2.8 Contributors & Users

2.8.1 The team

These are the people that created, that work or worked in thumbor across the releases:

Founders / Committers

- @heynemann - Founder and active committer
- @cezarsa - Committer
- @fabiomcosta - Founder and committer

Contributors

Contributors can be found in the Contributors page.

2.8.2 Who's using it



http://www.globo.com - globo.com uses thumbor to generate dynamic images through all products across the portal. Around 15 billion images served per month.



http://www.properati.com.ar/ - properati is also using thumbor to generate several different sizes of their properties photos, using smart cropping to get the best possible thumbnails.

> Thumbor made our lives better.
>
> At Properati.com.ar we care a lot about user experience.
>
> When our design team came up with a beautiful design that included thumbnails of 5 different sizes and specific cropping specs, instead of enhancing our home-made, simple thumbnail-generator process we moved to Thumbor and now, we cannot live without it.
>
> Thanks a lot!



http://yipit.com/ - yipit now uses thumbor behind the CloudFront CDN at Amazon. Their detailed experience with setting up thumbor can be seen at this blog post.

> Thumbor allows Yipit to iterate quickly on new designs without having to worry about introducing new image sizes.
>
> On demand image generation was just too slow when integrated into our application servers, but Thumbor makes it possible.

> No more going through old images and creating new thumbnails before we can roll out a new design!
>
> Our initial Thumbor installation took less than an hour to set up, and we haven't had to spend much time thinking about it since then.
>
> Zach Smith - CTO



http://oony.com is using thumbor to serve thumbnail images behind Amazon's Cloudfront CDN.

> We've previously adapted the size of the thumbnails to what was required by our design team, forcing us to have many different versions of the images we have on our site.
>
> With Thumbor we don't have to worry about this anymore, and we can quickly iterate and make changes to our layouts serving the optimal image format each time.
>
> Thumbor is awesome!



The Viadeo Group owns and operates professional social networks around the world with a total membership base of over 55 million professionals. Professionals use the networks to enhance their career prospects, discover business opportunities and build relationships with new contacts as well as to create effective online identities.

> With headquarters based in Paris, the Group currently has over 450 staff with offices and teams located in 10 countries.
>
> Viadeo is using Thumbor more and more. We used to have some home-made Java code to deliver images from http://www.viadeo.com. This code is still alive for some parts of our site.
>
> Since the end of summer 2013, Thumbor is a reality at Viadeo. First via IOS application for member's profile photos, then our news platform uses it for new parts of the site, taking more and more place and also some Android applications.
>
> Thumbor helps us in migrating and decoupling applications from our storage backend. We were able to move from NFS (centralized and very sensitive to high loads) to a distributed storage like HBase, using a hbase storage plugin. Using the same technique of lazy loading (via Storage cache in thumbor) we can imagine changing our image's storage at our convenience should apache HBase start to show deficiencies. This is really comfortable for Ops.

Threadless

TypeTees is an easy-to-use iPhone app that lets you speak your mind by putting your super witty slogan into an original tee and order it immediately.

> We use Thumbor to generate mobile thumbnails directly from the same large images that are sent to the t-shirt garment printer. It requires dealing with masks, feature trimming, transparent images, and replacing backgrounds to give users an easy-to-see preview of the t-shirt.
>
> Thumbor made this possible and simple without having to write an image processor from scrap.
>
> TypeTees was developed by www.prolificinteractive.com and you can learn more about how thumbor helped them at their engineering blog post.

Just Watch

At JustWatch, we're big fans of Thumbor as well.

We're serving it behind a CloudFront custom origin like many others, and features like WebP and smart cropping saved us huge amounts of time and bandwidth.



Ridelink

RideLink uses Thumbor to provide the most appropriate, and optimized, image for the platform our customer is using.

Setting it up and enhancing it was an easy task thanks to the good documentation and the available plugins.

Thanks to the team behind Thumbor for making our lifes easier!

Cheers! Erico Andrei - CTO



HeyCar

HeyCar is using Thumbor to optimize images for each customers device, taking network and screen sizes into account.

Our Thumbor instances run on our Kubernetes Cluster, served behind a CloudFront instance for caching purposes.

Huge Thanks to all Thumbor Contributors!

Marcelo Boeira - Software Engineer

Modalova

Modalova is an online shopping website dedicated to fashion for men and women.

We use Thumbor to generate product thumbnails on our website `Modalova`, on the grid and also on the Product Pages.

We serve more than 2,000,000 products to our customers everyday and work with more than 10,000 brands on the fashion Market.

Our Thumbor instances are running on Heroku, behind the CDN Cloudflare.

Thanks again for this wonderful project,

Cheers!

Gabriel Kaam - CEO & Founder

How to add my site or product here

If you are using thumbor and your site or product is not listed here, please create an issue and we'll include your logo and a short description on how you are using it here.

2.8.3 Hacking on Thumbor

So you want to contribute with thumbor? Welcome onboard!

There are a few things you'll need in order to properly start hacking on it.

First step is to fork it and create your own clone of thumbor.

Dependencies

The project uses uv to create the local `.venv`, install thumbor in editable mode, and keep dependency versions in sync with `uv.lock`. If uv is not already installed, `make setup` installs the pinned version into `.uv-venv`. On macOS, this workflow requires macOS 13 or newer.

You'll also need Python >= 3.11.

Other than that, you'll also need redis-server installed (for queued detector unit tests).

Initializing the Environment

Sync the development environment and compile the native extensions with:

```
$ make setup
$ make pre-commit
$ make compile_ext
```

The pre-commit hooks only need to be installed once per clone.

Running the Tests

Running the tests is as easy as:

```
$ make test
```

You should see the results of running your tests after an instant.

If you are experiencing "Too many open files" errors while running the tests, try increasing the number of open files per process, by running this command:

```
$ ulimit -S -n 2048
```

Read the file-limit discussion for more info on this.

Linting your code

Please ensure that your editor is configured to use black, flake8, isort, and pylint.

Even if that's the case, don't forget to run `make flake isort pylint` before committing and fixing any issues you find. That way you won't get a request for doing so in your PR.

Pull Requests

After hacking and testing your contribution, it is time to make a pull request. Make sure that your code is already integrated with the master branch of thumbor before asking for a pull request.

To add thumbor as a valid remote for your repository:

```
$ git remote add thumbor https://github.com/thumbor/thumbor.git
```

To merge thumbor's master with your fork:

```
$ git pull thumbor master
```

If there was anything to merge, just run your tests again. If they pass, send a pull request.

Introducing a new Dependency

If we introduce a new dependency, the testing docker images need to be updated.

If the new dependency requires changes to the docker image, make sure to update the TestDockerfile file.

Then build and publish with:

```
make test-docker-build test-docker-publish
```

Remember that you must be logged in with your Docker Hub account and you must be part of the thumbororg team of administrators.

Running tests in docker

If you do not wish to configure your environment with thumbor's dependencies, you can build local Docker images and run the tests with:

```
make test-docker-run
```

Or if you want to run a specific python version with your tests:

```
make test-docker-311-run
```

These targets build the corresponding local test image before running it.

Just replace '311' with the python version you want: 311, 312, 313, 314 or 315.

2.8.4 Licensing

Thumbor is licensed under the MIT License:

The MIT License

Copyright (c) 2011 globo.com thumbor@googlegroups.com

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p
piexif, 99

INDEX

M

module
    piexif, 99

P

piexif
    module, 99