

Image Factory

- [What is the image factory?](#)
- [How do I use the image factory?](#)

What is the image factory?

The Image Factory is a tool for generating optimised, resized versions of images on demand. It allows you to define the dimensions and apply transformations such as cropping, resizing, format conversion, and quality adjustments, while generating a signed URL for the resulting image.

Introduction

The image factory can be called using the `image_factory()` helper function from within PHP (.php and .blade.php) and Twig (.twig) files. When initializing an instance, the width and height of the resized image must be provided (in pixels).

The image factory outputs a signed URL to the resized image with an embedded validation token. This prevents being able to generate an infinite number of uniquely configured URLs that would drain the resources on the server.

PHP

```
image_factory(600, 800);  
// outputs: https://store.com/image-factory/640fa397eb07791f~600x800  
  
image_factory(600, 800)->path('/images/example.jpg');  
// outputs: https://store.com/image-factory/640fa397eb07791f~600x800/images/example.jpg  
  
image_factory(600, 800)->path('/images/example.jpg')->retina();  
// outputs: https://store.com/image-factory/632ab6d293827ff6~600x800@2x/images/example.jpg
```

Twig

```
{{ image_factory(600, 800) }}  
#{ outputs: https://store.com/image-factory/640fa397eb07791f~600x800 #}  
  
{{ image_factory(600, 800).path('/images/example.jpg') }}
```

```
{# outputs: https://store.com/image-factory/640fa397eb07791f~600x800/images/example.jpg #}

{{ image_factory(600, 800).path('/images/example.jpg').retina() }}

{# outputs: https://store.com/image-factory/632ab6d293827ff6~600x800@2x/images/example.jpg #}
```

Twig / Vue

If using the Aero component system (Vue.js template within a .twig file), the image URL can be built by merging the image factory prefix with the JavaScript variable:

```
<div v-if="image">
  <picture>
    <source :srcset="'{{ image_factory(600, 600) }}/' + image.file + '.webp 1x,{{
image_factory(600, 800).retina() }}/' + image.file + '.webp 2x'"
    type="image/webp">
    
  </picture>
</div>
```

Available Options

There are several options that can be used to manipulate the image. These can be chained.

path(\$path)

The `path` option is used to specify the path to the original image within the `storage/app/public` directory.

Twig:

```
{% set image = listing.images | first %}


```

PHP:

```
$image = image_factory(150, 300)->path($path);
```

Alternatively, the path can be provided as the third parameter in the initialisation of the instance:

Twig:

```
{% set image = listing.images | first %}  

```

PHP:

```
$image = image_factory(150, 300, $path);
```

webp()

The `webp` option is used to serve a `.webp` (a modern lossless compression) image.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->webp();
```

retina()

The `retina` option is used to serve an image that is twice the size (in pixels) of the designated dimensions.

For example, an image generated from `image_factory(500, 500)->retina()` would result in an image with dimensions 1000px by 1000px.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->retina();
```

dpr(\$scale)

The `dpr` option is used to scale the image dimensions.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->dpr(3.5);
```

flip()

The `flip` option transforms the image by flipping the x-axis.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->flip();
```

contain()

The `contain` option keeps the resized image within the dimensional constraints whilst retaining the original aspect ratio.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->contain();
```

crop()

The `crop` option scales the image to the dimensional constraints and removes any overflowing image from the canvas.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->crop();
```

`rotate($angle)`

The `rotate` option rotates the image by the given angle (in degrees counter-clockwise).

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->rotate(90);
```

`sharpen($amount)`

The `sharpen` option sharpens the image by the given amount (0 - 100).

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->sharpen(25);
```

`pixelate($size)`

The `pixelate` option applies a pixelation effect to the image with a given size of pixels.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->pixelate(5);
```

quality(\$percentage)

The `quality` option sets the quality of the outputted image. The default is 80%.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->quality(95);
```

invert()

The `invert` option reverses all colors in the image.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->invert();
```

greyscale()

The `greyscale` option turns the image into a greyscale version.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->greyscale();
```

placeholder(\$values)

The `placeholder` option returns a placeholder string.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->placeholder('#ff0000');
```

By default, this is an encoded inline SVG path, which can be changed by setting a custom resolver:

```
\Aero\Store\Services\ImageFactory::setPlaceholderResolver(function ($image) {  
    return asset('placeholder.svg');  
});
```

If using a custom resolver, the values passed into the placeholder can then be accessed using the `$placeholderValues` property:

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->placeholder(['type' => 'product', 'category' => 'dress']);
```

```
\Aero\Store\Services\ImageFactory::setPlaceholderResolver(function ($image) {  
    return asset("placeholder-{$image->placeholderValues['type']}.svg#{$image->placeholderValues['category']}");  
});
```

options(\$options)

The `options` option allows multiple options to be specified by calling one method. The individual options can either be passed as separate arguments or as an array.

Twig:

```


// or ...


```

PHP:

```
$image = image_factory(150, 300)->options('crop', ['pixelate' => 100], ['quality' => 30]);

// or ...

$image = image_factory(150, 300)->options(['crop', ['pixelate' => 100], ['quality' => 30]]);
```

How do I use the image factory?

The image factory can be called using the `image_factory()` helper function from within PHP (.php and .blade.php) and Twig (.twig) files. When initialising an instance, the width and height of the resized image must be provided (in pixels).

The image factory outputs a signed URL to the resized image with an embedded validation token. This prevents being able to generate an infinite number of uniquely configured URLs that would drain the resources on the server.

PHP

```
image_factory(600, 800);  
// outputs: https://store.com/image-factory/640fa397eb07791f~600x800  
  
image_factory(600, 800)->path('/images/example.jpg');  
// outputs: https://store.com/image-factory/640fa397eb07791f~600x800/images/example.jpg  
  
image_factory(600, 800)->path('/images/example.jpg')->retina();  
// outputs: https://store.com/image-factory/632ab6d293827ff6~600x800@2x/images/example.jpg
```

Twig

```
{{ image_factory(600, 800) }}  
{# outputs: https://store.com/image-factory/640fa397eb07791f~600x800 #}  
  
{{ image_factory(600, 800).path('/images/example.jpg') }}  
{# outputs: https://store.com/image-factory/640fa397eb07791f~600x800/images/example.jpg #}  
  
{{ image_factory(600, 800).path('/images/example.jpg').retina() }}  
{# outputs: https://store.com/image-factory/632ab6d293827ff6~600x800@2x/images/example.jpg #}
```

Twig / Vue

If using the Aero component system (Vue.js template within a .twig file), the image URL can be built by merging the image factory prefix with the JavaScript variable:

```
<div v-if="image">
  <picture>
    <source :srcset="'{{ image_factory(600, 600) }}/' + image.file + '.webp 1x,{{
image_factory(600, 800).retina() }}/' + image.file + '.webp 2x'"
    type="image/webp">
    
  </picture>
</div>
```