TileServer GL Documentation

Release 1.0

MapTiler.com

May 17, 2024

CONTENTS

Contents:

CHAPTER

ONE

INSTALLATION

1.1 Docker

When running docker image, no special installation is needed – the docker will automatically download the image if not present.

Just run `docker run --rm -it -v $(pwd):/data -p 8080:8080 maptiler/tileserver-gl`.

Additional options (see *Usage*) can be passed to the TileServer GL by appending them to the end of this command. You can, for example, do the following:

- `docker run ... maptiler/tileserver-gl --file my-tiles.mbtiles` – explicitly specify which mbtiles to use (if you have more in the folder)
- `docker run ... maptiler/tileserver-gl --verbose` – to see the default config created automatically

1.2 npm

npm is supported on the following platforms with *Native Dependencies* installed.

- Operating systems:
  - Ubuntu 22.04 (x64/arm64)
  - macOS 12 (x64/arm64)
  - Windows (x64)
- Node.js 18,20

1.2.1 Install globally from npmjs.

```
npm install -g tileserver-gl
tileserver-gl
```

1.2.2 Install locally from source

```
git clone https://github.com/maptiler/tileserver-gl.git
cd tileserver-gl
npm install
node .
```

1.2.3 Native dependencies

Ubuntu 22.04 (x64/arm64)

- apt install build-essential pkg-config xvfb libglfw3-dev libuv1-dev libjpeg-turbo8 libicu70 libcairo2-dev libpango1.0-dev libjpeg-dev libgif-dev librsvg2-dev gir1.2-rsvg-2.0 librsvg2-2 librsvg2-common libcurl4-openssl-dev libpixman-1-dev libpixman-1-0

MacOS 12 (x64/arm64)

- brew install pkg-config cairo libpng jpeg giflib

Windows (x64)

- Microsoft Visual C++ 2015-2022 Redistributable

1.3 `tileserver-gl-light` on npm

Alternatively, you can use `tileserver-gl-light` package instead, which is pure javascript (does not have any native dependencies) and can run anywhere, but does not contain rasterization features.

CHAPTER

TWO

USAGE

2.1 Getting started

```
Usage: main.js tileserver-gl [file] [options]

Options:
  --file <file>             MBTiles or PMTiles file
                              ignored if the configuration file is also specified
  --mbtiles <file>          (DEPRECIATED) MBTiles file
                              ignored if file is also specified
                              ignored if the configuration file is also specified
  -c, --config <file>       Configuration file [config.json] (default: "config.json")
  -b, --bind <address>      Bind address
  -p, --port <port>         Port [8080] (default: 8080)
  -C|--no-cors              Disable Cross-origin resource sharing headers
  -u|--public_url <url>     Enable exposing the server on subpaths, not necessarily the␣
↪root of the domain
  -V, --verbose             More verbose output
  -s, --silent              Less verbose output
  -l|--log_file <file>      output log file (defaults to standard out)
  -f|--log_format <format>  define the log format:  https://github.com/expressjs/morgan
↪#morganformat-options
  -v, --version             output the version number
  -h, --help                display help for command
```

2.2 Default preview style and configuration

- If no configuration file is specified, a default preview style (compatible with openmaptiles) is used.
- If no data file is specified (and is not found in the current working directory), a sample file is downloaded (showing the Zurich area)

2.3 Reloading the configuration

It is possible to reload the configuration file without restarting the whole process by sending a SIGHUP signal to the node process.

- The *docker kill -s HUP tileserver-gl* command can be used when running the tileserver-gl docker container.
- The *docker-compose kill -s HUP tileserver-gl-service-name* can be used when tileserver-gl is run as a docker-compose service.

2.4 Docker and *–port*

When running tileserver-gl in a Docker container, using the *–port* option would make the container incorrectly seem unhealthy. Instead, it is advised to use Docker's port mapping and map the default port 8080 to the desired external port.

CHAPTER

THREE

CONFIGURATION FILE

The configuration file defines the behavior of the application. It's a regular JSON file.

Example:

```
{
  "options": {
    "paths": {
      "root": "",
      "fonts": "fonts",
      "sprites": "sprites",
      "icons": "icons",
      "styles": "styles",
      "mbtiles": "data",
      "pmtiles": "data"
    },
    "domains": [
      "localhost:8080",
      "127.0.0.1:8080"
    ],
    "formatQuality": {
      "jpeg": 80,
      "webp": 90
    },
    "maxScaleFactor": 3,
    "maxSize": 2048,
    "pbfAlias": "pbf",
    "serveAllFonts": false,
    "serveAllStyles": false,
    "serveStaticMaps": true,
    "allowRemoteMarkerIcons": true,
    "allowInlineMarkerImages": true,
    "staticAttributionText": "© OpenMapTiles  © OpenStreetMaps",
    "tileMargin": 0
  },
  "styles": {
    "basic": {
      "style": "basic.json",
      "tilejson": {
        "type": "overlay",
        "bounds": [8.44806, 47.32023, 8.62537, 47.43468]
      }
```

(continues on next page)

(continued from previous page)

```
    },
    "hybrid": {
      "style": "satellite-hybrid.json",
      "serve_rendered": false,
      "tilejson": {
        "format": "webp"
      }
    }
  },
  "data": {
    "zurich-vector": {
      "mbtiles": "zurich.mbtiles"
    }
  }
}
```

3.1 options

3.1.1 paths

Defines where to look for the different types of input data.

The value of `root` is used as prefix for all data types.

3.1.2 domains

You can use this to optionally specify on what domains the rendered tiles are accessible. This can be used for basic load-balancing or to bypass browser's limit for the number of connections per domain.

3.1.3 frontPage

Path to the html (relative to `root` path) to use as a front page.

Use `true` (or nothing) to serve the default TileServer GL front page with list of styles and data. Use `false` to disable the front page altogether (404).

3.1.4 formatQuality

Quality of the compression of individual image formats. [0-100]

3.1.5 maxScaleFactor

Maximum scale factor to allow in raster tile and static maps requests (e.g. `@3x` suffix). Also see `maxSize` below. Default value is `3`, maximum `9`.

3.1.6 maxSize

Maximum image side length to be allowed to be rendered (including scale factor). Be careful when changing this value since there are hardware limits that need to be considered. Default is `2048`.

3.1.7 tileMargin

Additional image side length added during tile rendering that is cropped from the delivered tile. This is useful for resolving the issue with cropped labels, but it does come with a performance degradation, because additional, adjacent vector tiles need to be loaded to generate a single tile. Default is `0` to disable this processing.

3.1.8 minRendererPoolSizes

Minimum amount of raster tile renderers per scale factor. The value is an array: the first element is the minimum amount of renderers for scale factor one, the second for scale factor two and so on. If the array has less elements than `maxScaleFactor`, then the last element is used for all remaining scale factors as well. Selecting renderer pool sizes is a trade-off between memory use and speed. A reasonable value will depend on your hardware and your amount of styles and scale factors. If you have plenty of memory, you'll want to set this equal to `maxRendererPoolSizes` to avoid increased latency due to renderer destruction and recreation. If you need to conserve memory, you'll want something lower than `maxRendererPoolSizes`, possibly allocating more renderers to scale factors that are more common. Default is `[8, 4, 2]`.

3.1.9 maxRendererPoolSizes

Maximum amount of raster tile renderers per scale factor. The value and considerations are similar to `minRendererPoolSizes` above. If you have plenty of memory, try setting these equal to or slightly above your processor count, e.g. if you have four processors, try a value of `[6]`. If you need to conserve memory, try lower values for scale factors that are less common. Default is `[16, 8, 4]`.

3.1.10 pbfAlias

Some CDNs did not handle .pbf extension as a static file correctly. The default URLs (with .pbf) are always available, but an alternative can be set. An example extension suffix would be ".pbf.pict".

3.1.11 serveAllFonts

If this option is enabled, all the fonts from the `paths.fonts` will be served. Otherwise only the fonts referenced by available styles will be served.

3.1.12 serveAllStyles

If this option is enabled, all the styles from the `paths.styles` will be served. (No recursion, only `.json` files are used.) The process will also watch for changes in this directory and remove/add more styles dynamically. It is recommended to also use the `serveAllFonts` option when using this option.

3.1.13 serveStaticMaps

If this option is enabled, all the static map endpoints will be served. Default is `true`.

3.1.14 watermark

Optional string to be rendered into the raster tiles and static maps as watermark (bottom-left corner). Not used by default.

3.1.15 staticAttributionText

Optional string to be rendered in the static images endpoint. Text will be rendered in the bottom-right corner, and styled similar to attribution on web-based maps (text only, links not supported). Not used by default.

3.1.16 allowRemoteMarkerIcons

Allows the rendering of marker icons fetched via http(s) hyperlinks. For security reasons only allow this if you can control the origins from where the markers are fetched! Default is to disallow fetching of icons from remote sources.

3.1.17 allowInlineMarkerImages

Allows the rendering of inline marker icons or base64 urls. For security reasons only allow this if you can control the origins from where the markers are fetched! Not used by default.

3.2 styles

Each item in this object defines one style (map). It can have the following options:

- `style` – name of the style json file [required]
- `serve_rendered` – whether to render the raster tiles for this style or not
- `serve_data` – whether to allow access to the original tiles, sprites and required glyphs
- `tilejson` – properties to add to the TileJSON created for the raster data
  - `format` and `bounds` can be especially useful

3.3 data

Each item specifies one data source which should be made accessible by the server. It has to have one of the following options:

- `mbtiles` – name of the mbtiles file
- `pmtiles` – name of the pmtiles file or url.

For example:

```
"data": {
  "source1": {
    "mbtiles": "source1.mbtiles"
  },
  "source2": {
    "pmtiles": "source2.pmtiles"
  },
  "source3": {
    "pmtiles": "https://foo.lan/source3.pmtiles"
  }
}
```

The data source does not need to be specified here unless you explicitly want to serve the raw data.

3.4 Referencing local files from style JSON

You can link various data sources from the style JSON (for example even remote TileJSONs).

3.4.1 MBTiles

To specify that you want to use local mbtiles, use to following syntax: `mbtiles://source1.mbtiles`. TileServer-GL will try to find the file `source1.mbtiles` in `root` + `mbtiles` path.

For example:

```
"sources": {
  "source1": {
    "url": "mbtiles://source1.mbtiles",
    "type": "vector"
  }
}
```

Alternatively, you can use `mbtiles://{source1}` to reference existing data object from the config. In this case, the server will look into the `config.json` to determine what file to use by data id. For the config above, this is equivalent to `mbtiles://source1.mbtiles`.

3.4.2 PMTiles

To specify that you want to use local pmtiles, use to following syntax: `pmtiles://source2.pmtiles`. TileServer-GL will try to find the file `source2.pmtiles` in `root` + `pmtiles` path.

To specify that you want to use a url based pmtiles, use to following syntax: `pmtiles://https://foo.lan/source3.pmtiles`.

For example:

```
"sources": {
  "source2": {
    "url": "pmtiles://source2.pmtiles",
    "type": "vector"
  },
  "source3": {
    "url": "pmtiles://https://foo.lan/source3.pmtiles",
    "type": "vector"
  },
}
```

Alternatively, you can use `pmtiles://{source2}` to reference existing data object from the config. In this case, the server will look into the `config.json` to determine what file to use by data id. For the config above, this is equivalent to `pmtiles://source2.mbtiles`.

3.4.3 Sprites

If your style requires any sprites, make sure the style JSON contains proper path in the `sprite` property.

It can be a local path (e.g. `my-style/sprite`) or remote http(s) location (e.g. `https://mycdn.com/my-style/sprite`). Several possible extension are added to this path, so the following files should be present:

- `sprite.json`
- `sprite.png`
- `sprite@2x.json`
- `sprite@2x.png`

You can also use the following placeholders in the sprite path for easier use:

- `{style}` – gets replaced with the name of the style file (`xxx.json`)
- `{styleJsonFolder}` – gets replaced with the path to the style file

3.4.4 Fonts (glyphs)

Similarly to the sprites, the style JSON also needs to contain proper paths to the font glyphs (in the `glyphs` property) and can be both local and remote.

It should contain the following placeholders:

- `{fontstack}` – name of the font and variant
- `{range}` – range of the glyphs

For example `"glyphs": "{fontstack}/{range}.pbf"` will instruct TileServer-GL to look for the files such as `fonts/Open Sans/0-255.pbf` (`fonts` come from the `paths` property of the `config.json` example above).

CHAPTER

FOUR

DEPLOYMENT

Typically, you should use nginx, lighttpd or apache on the frontend. The tileserver-gl server is hidden behind it in production deployment.

4.1 Caching

There is a plenty of options you can use to create proper caching infrastructure: Varnish, Cloudflare, ...

4.1.1 Cloudflare Cache Rules

Cloudflare supports custom rules for configuring caching: https://developers.cloudflare.com/cache/about/cache-rules/

tileserver-gl renders tiles in multiple formats - `.png`, `.jpg (jpeg)`, `.webp` for the raster endpoints, `.pbf` for vector endpoint. In addition, style information is generated with `.json` format.

Endpoint data can be configured to be cached by Cloudflare. For example to cache vector endpoint you will need to configure Cloudflare rules for the `.pbf` and `.json` data.

Create a rule which matches `hostname (equal)` and `URI Path (ends with)` for `.pbf` and `.json` fields. Set cache status to eligible for cache to enable the caching and overwrite the `Edge TTL` with `Browser TTL` to be 7 days (depends on your application usage).

This will ensure that your tiles are cached on the client side and by Cloudflare for seven days. If the tileserver is down or user has no internet access it will try to use cached tiles.

Note that `Browser TTL` will overwrite expiration dates on the client device. If you rebuild your maps, old tiles will be rendered until it expires or cache is cleared on the client device.

4.1.2 Nginx Cache

If you have a reverse proxy setup in front of the tileserver you may want to enable caching as it will greatly offload requests from the application.

Configure the proxy cache path directive to initialize your cache store:

```
proxy_cache_path /var/cache/nginx/tileserver
                 keys_zone=TileserverCache:50m
                 levels=1:2
                 inactive=2w
                 max_size=10g;
```

Make sure to give proper permissions for the /var/cache/nginx/tileserver folder. Usually nginx is running with www-data user. Enable caching on specific proxy pass:

```
location / {
  include proxy_params;
  proxy_pass http://127.0.0.1:8080/;

  proxy_cache TileserverCache;
  proxy_cache_valid 200 1w;

  # add_header X-Cache-Status $upstream_cache_status;
}
```

If you need to confirm whether caching works or not, uncomment the X-Cache-Status header. This will return a header on response with *HIT* or *MISS* header value which indicates if nginx cached the response or not.

Make sure to clean your cache by removing files in the configured directory after you change your styles or tile information. You may experiment with the caching values to fit your needs.

More about Nginx caching: https://docs.nginx.com/nginx/admin-guide/content-cache/content-caching/

4.2 Securing

Nginx can be used to add protection via https, password, referrer, IP address restriction, access keys, etc.

4.3 Running behind a proxy or a load-balancer

If you need to run TileServer GL behind a proxy, make sure the proxy sends `X-Forwarded-*` headers to the server (most importantly `X-Forwarded-Host` and `X-Forwarded-Proto`) to ensure the URLs generated inside TileJSON, etc. are using the desired domain and protocol.

4.3.1 Nginx Reverse Proxy

An example nginx reverse proxy server configuration for HTTPS connections. It enables caching, CORS and Cloudflare Authenticated Pulls.

```
proxy_cache_path /var/cache/nginx/tileserver
                 keys_zone=TileserverCache:50m
                 levels=1:2
                 inactive=2w
                 max_size=1g;

map_hash_bucket_size 128;
map $http_origin $allow_origin {
    https://www.example.com $http_origin;
    default "";
}

server {
  listen 443 ssl http2;
```

(continues on next page)

(continued from previous page)

```
  listen [::]:443 ssl http2;

  ssl_certificate         /etc/ssl/www.example.com/cert.pem;
  ssl_certificate_key     /etc/ssl/www.example.com/key.pem;

  # https://developers.cloudflare.com/ssl/origin-configuration/authenticated-origin-pull/
  ssl_client_certificate  /etc/ssl/cloudflare.pem;
  ssl_verify_client on;

  server_name www.example.com example.com;

  # Disable root application access. You may want to allow this in development.
  location ~ ^/$ {
    return 404;
  }

  # Disable root application access. You may want to allow this in development.
  location /favicon.ico {
    return 404;
  }

  location / {
    # This include directive sets up required headers for proxy and proxy cache.
    # As well it includes the required ``X-Forwarded-*`` headers for tileserver to␣
↪properly generate tiles.
    include proxy_params;

    proxy_pass http://127.0.0.1:8080/;

    # Disable default CORS headers
    proxy_hide_header Access-Control-Allow-Origin;

    # Enable proxy cache
    proxy_cache TileserverCache;
    proxy_cache_valid 200 1w;

    # Set our custom CORS
    add_header 'Access-Control-Allow-Origin' $allow_origin;

    # If you need to see nginx cache status. Uncomment line below.
    # add_header X-Cache-Status $upstream_cache_status;
  }
}
```

CHAPTER

FIVE

AVAILABLE ENDPOINTS

If you visit the server on the configured port (default 8080) you can see your maps appearing in the browser.

5.1 Styles

- Styles are served at `/styles/{id}/style.json` (+ array at `/styles.json`)
  - Sprites at `/styles/{id}/sprite[/spriteID][@2x].{format}`
  - Fonts at `/fonts/{fontstack}/{start}-{end}.pbf`

5.2 Rendered tiles

- Rendered tiles are served at `/styles/{id}[/{tileSize}]/{z}/{x}/{y}[@2x].{format}`
  - The optional ratio `@2x` (ex. `@2x`, `@3x`, `@4x`) part can be used to render HiDPI (retina) tiles
  - The optional tile size `/{tileSize}` (ex. `/256`, `/512`). if omitted, tileSize defaults to 256.
  - Available formats: `png`, `jpg` (`jpeg`), `webp`
  - TileJSON at `/styles[/{tileSize}]/{id}.json`
- The rendered tiles are not available in the `tileserver-gl-light` version.

5.3 WMTS Capabilities

- WMTS Capabilities are served at `/styles/{id}/wmts.xml`

5.4 Static images

- Several endpoints:
  - `/styles/{id}/static/{lon},{lat},{zoom}[@{bearing}[,{pitch}]]/{width}x{height}[@2x].{format}` (center-based)
  - `/styles/{id}/static/{minx},{miny},{maxx},{maxy}/{width}x{height}[@2x].{format}` (area-based)
  - `/styles/{id}/static/auto/{width}x{height}[@2x].{format}` (autofit path – see below)

- All the static image endpoints additionally support following query parameters:
  - `path - ((fill|stroke|width)\:[^\|]+\|)*(enc:.+|-?\d+(\.\d*)?,-?\d+(\.\d*)?(\|-?\d+(\.\d*)?,-?\d+(\.\d*)?)+)`
    - comma-separated `lng,lat`, pipe-separated pairs
      - e.g. `path=5.9,45.8|5.9,47.8|10.5,47.8|10.5,45.8|5.9,45.8`
    - Google Encoded Polyline Format
      - e.g. `path=enc:_p~iF~ps|U_ulLnnqC_mqNvxq`@`
      - If 'enc:' is used, the rest of the path parameter is considered to be part of the encoded polyline string – do not specify the coordinate pairs.
    - With options (fill|stroke|width)
      - e.g. `path=stroke:yellow|width:2|fill:green|5.9,45.8|5.9,47.8|10.5,47.8|10.5,45.8|5.9,45.8` or `path=stroke:blue|width:1|fill:yellow|enc:_p~iF~ps|U_ulLnnqC_mqNvxq`@`
    - can be provided multiple times
  - `latlng` - indicates coordinates are in `lat,lng` order rather than the usual `lng,lat` for paths and markers
  - `fill` - default color to use as the fill (e.g. `red`, `rgba(255,255,255,0.5)`, `#0000ff`) for all paths
  - `stroke` - default color of the path stroke for all paths
  - `width` - default width of the stroke for all paths
  - `linecap` - rendering style for the start and end points of all paths - see https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/lineCap
  - `linejoin` - rendering style for joining successive segments of all paths when the direction changes - see https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/lineJoin
  - `border` - color of the optional border stroke for all paths ; the border is like a halo around the stroke
  - `borderwidth` - width of the border stroke (default 10% of stroke width) for all paths
  - `marker` - Marker in format `lng,lat|iconPath|option|option|...`
    - Will be rendered with the bottom center at the provided location
    - `lng,lat` and `iconPath` are mandatory and icons won't be rendered without them
    - `iconPath` is either a link to an image served via http(s) or a path to a file relative to the configured icon path
    - `option` must adhere to the format `optionName:optionValue` and supports the following names
      - `scale` - Factor to scale image by
      - e.g. `0.5` - Scales the image to half it's original size
      - `offset` - Image offset as positive or negative pixel value in format `[offsetX],[offsetY]`
      - scales with `scale` parameter since image placement is relative to it's size
      - e.g. `2,-4` - Image will be moved 2 pixel to the right and 4 pixel in the upwards direction from the provided location
    - e.g. `5.9,45.8|marker-start.svg|scale:0.5|offset:2,-4`
    - can be provided multiple times
  - `padding` - "percentage" padding for fitted endpoints (area-based and path autofit)

        * value of `0.1` means "add 10% size to each side to make sure the area of interest is nicely visible"
    - `maxzoom` - Maximum zoom level (only for auto endpoint where zoom level is calculated and not provided)
- You can also use (experimental) `/styles/{id}/static/raw/...` endpoints with raw spherical mercator coordinates (EPSG:3857) instead of WGS84.
- The static images are not available in the `tileserver-gl-light` version.

5.5 Source data

- Source data are served at `/data/{id}/{z}/{x}/{y}.{format}`
    - Format depends on the source file (usually `png` or `pbf`)
        * `geojson` is also available (useful for inspecting the tiles) in case the original format is `pbf`
    - TileJSON at `/data/{id}.json`

5.6 TileJSON arrays

Array of all TileJSONs is at `[/{tileSize}]/index.json` (`[/{tileSize}]/rendered.json`; `/data.json`)

- The optional tile size `/{tileSize}` (ex. `/256`, `/512`). if omitted, tileSize defaults to 256.

5.7 List of available fonts

Array of names of the available fonts is at `/fonts.json`

5.8 Health check

Endpoint reporting health status is at `/health` and currently returns:

- `503` Starting - for a short period before everything is initialized
- `200` OK - when the server is running

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search