+++ title = “Image rendering” description = “Image rendering” keywords = [“grafana”, “image”, “rendering”, “plugin”] aliases = [“/docs/grafana/latest/administration/image_rendering/”] weight = 55 +++

Image rendering

Grafana supports automatic rendering of panels as PNG images. This allows Grafana to automatically generate images of your panels to include in [alert notifications]({{< relref “../alerting/old-alerting/notifications.md” >}}), [PDF export]({{< relref “../enterprise/export-pdf.md” >}}), and [Reporting]({{< relref “../enterprise/reporting.md” >}}). PDF Export and Reporting are available only in [Grafana Enterprise]({{< relref “../enterprise/” >}}).

> **Note:** Image rendering of dashboards is not supported at this time.

While an image is being rendered, the PNG image is temporarily written to the file system. When the image is rendered, the PNG image is temporarily written to the `png` folder in the Grafana `data` folder.

A background job runs every 10 minutes and removes temporary images. You can configure how long an image should be stored before being removed by configuring the [temp_data_lifetime]({{< relref “../administration/configuration/#temp_data_lifetime” >}}) setting.

You can also render a PNG by clicking the dropdown arrow next to a panel title, then clicking **Share > Direct link rendered image**.

Alerting and render limits

Alert notifications can include images, but rendering many images at the same time can overload the server where the renderer is running. For instructions of how to configure this, see [concurrent_render_limit]({{< relref “../administration/configuration/#concurrent_render_limit” >}}).

Install Grafana Image Renderer plugin

> **Note:** Starting from Grafana v7.0.0, all PhantomJS support has been removed. Please use the Grafana Image Renderer plugin or remote rendering service.

To install the plugin, refer to the Grafana Image Renderer Installation instructions.

Configuration

The Grafana Image Renderer plugin has a number of configuration options that are used in plugin or remote rendering modes.

In plugin mode, you can specify them directly in the [Grafana configuration file]({{< relref "../administration/configuration/#plugingrafana-image-renderer" >}}).

In remote rendering mode, you can specify them in a `.json` configuration file or, for some of them, you can override the configuration defaults using environment variables.

Configuration file

You can update your settings by using a configuration file, see default.json for defaults. Note that any configured environment variable takes precedence over configuration file settings.

You can volume mount your custom configuration file when starting the docker container:

```
docker run -d --name=renderer --network=host -v /some/path/config.json:/usr/src/app/config.j
```

You can see a docker-compose example using a custom configuration file here.

Rendering mode

You can instruct how headless browser instances are created by configuring a rendering mode. Default is `default`, other supported values are `clustered` and `reusable`.

Default Default mode will create a new browser instance on each request. When handling multiple concurrent requests, this mode increases memory usage as it will launch multiple browsers at the same time. If you want to set a maximum number of browser to open, you'll need to use the clustered mode.

> **Note:** When using the `default` mode, it's recommended to not remove the default Chromium flag `--disable-gpu`. When receiving a lot of concurrent requests, not using this flag can cause Puppeteer `newPage` function to freeze, causing request timeouts and leaving browsers open.

```
RENDERING_MODE=default
```

```
{
  "rendering": {
    "mode": "default"
  }
}
```

Clustered With the `clustered` mode, you can configure how many browser instances or incognito pages can execute concurrently. Default is `browser` and will ensure a maximum amount of browser instances can execute concurrently.

Mode `context` will ensure a maximum amount of incognito pages can execute concurrently. You can also configure the maximum concurrency allowed, which per default is `5`, and the maximum duration of a rendering request, which per default is `30` seconds.

Using a cluster of incognito pages is more performant and consumes less CPU and memory than a cluster of browsers. However, if one page crashes it can bring down the entire browser with it (making all the rendering requests happening at the same time fail). Also, each page isn't guaranteed to be totally clean (cookies and storage might bleed-through as seen here).

```
RENDERING_MODE=clustered
RENDERING_CLUSTERING_MODE=browser
RENDERING_CLUSTERING_MAX_CONCURRENCY=5
RENDERING_CLUSTERING_TIMEOUT=30
```

```
{
  "rendering": {
    "mode": "clustered",
    "clustering": {
      "mode": "browser",
      "maxConcurrency": 5,
      "timeout": 30
    }
  }
}
```

Reusable (experimental) When using the rendering mode `reusable`, one browser instance will be created and reused. A new incognito page will be opened for each request. This mode is experimental since, if the browser instance crashes, it will not automatically be restarted. You can achieve a similar behavior using `clustered` mode with a high `maxConcurrency` setting.

```
RENDERING_MODE=reusable
```

```
{
  "rendering": {
    "mode": "reusable"
  }
}
```

Optimize the performance, CPU and memory usage of the image renderer The performance and resources consumption of the different modes depend a lot on the number of concurrent requests your service is handling. To understand how many concurrent requests your service is handling, [monitor your image renderer service]({{< relref "./monitoring/" >}}).

With no concurrent requests, the different modes show very similar performance and CPU / memory usage.

When handling concurrent requests, we see the following trends:

- To improve performance and reduce CPU and memory consumption, use clustered mode with `RENDERING_CLUSTERING_MODE` set as `context`. This parallelizes incognito pages instead of browsers.
- If you use the clustered mode with a `maxConcurrency` setting below your average number of concurrent requests, performance will drop as the rendering requests will need to wait for the other to finish before getting access to an incognito page / browser.

To achieve better performance, monitor the machine on which your service is running. If you don't have enough memory and / or CPU, every rendering step will be slower than usual, increasing the duration of every rendering request.

Other available settings

> **Note:** Please note that not all settings are available using environment variables. If there is no example using environment variable below, it means that you need to update the configuration file.

HTTP host Change the listening host of the HTTP server. Default is unset and will use the local host.

```
HTTP_HOST=localhost
```

```
{
  "service": {
    "host": "localhost"
  }
}
```

HTTP port Change the listening port of the HTTP server. Default is `8081`. Setting `0` will automatically assign a port not in use.

```
HTTP_PORT=0
```

```
{
  "service": {
    "port": 0
  }
}
```

Enable Prometheus metrics You can enable Prometheus metrics endpoint `/metrics` using the environment variable `ENABLE_METRICS`. Node.js and render request duration metrics are included, see output example for details.

Default is `false`.

```
ENABLE_METRICS=true
```

```
{
  "service": {
    "metrics": {
      "enabled": true,
      "collectDefaultMetrics": true,
      "requestDurationBuckets": [1, 5, 7, 9, 11, 13, 15, 20, 30]
    }
  }
}
```

Log level Change the log level. Default is `info` and will include log messages with level `error`, `warning` and `info`.

```
LOG_LEVEL=debug
```

```
{
  "service": {
    "logging": {
      "level": "debug",
      "console": {
        "json": false,
        "colorize": true
      }
    }
  }
}
```

Verbose logging Instruct headless browser instance whether to capture and log verbose information when rendering an image. Default is `false` and will only capture and log error messages. When enabled (`true`) debug messages are captured and logged as well.

Note that you need to change log level to `debug`, see above, for the verbose information to be included in the logs.

```
RENDERING_VERBOSE_LOGGING=true
```

```
{
  "rendering": {
    "verboseLogging": true
  }
}
```

Capture browser output Instruct headless browser instance whether to output its debug and error messages into running process of remote rendering service. Default is `false`. This can be useful to enable (`true`) when troubleshooting.

```
RENDERING_DUMPIO=true
```

```
{
  "rendering": {
    "dumpio": true
  }
}
```

Custom Chrome/Chromium If you already have Chrome or Chromium installed on your system, then you can use this instead of the pre-packaged version of Chromium.

> **Note:** Please note that this is not recommended, since you may encounter problems if the installed version of Chrome/Chromium is not compatible with the Grafana Image renderer plugin.

You need to make sure that the Chrome/Chromium executable is available for the Grafana/image rendering service process.

```
CHROME_BIN="/usr/bin/chromium-browser"
```

```
{
  "rendering": {
    "chromeBin": "/usr/bin/chromium-browser"
  }
}
```

Start browser with additional arguments Additional arguments to pass to the headless browser instance. Defaults are `--no-sandbox,--disable-gpu`. The list of Chromium flags can be found here and the list of flags used as defaults by Puppeteer can be found there. Multiple arguments is separated with comma-character.

```
RENDERING_ARGS=--no-sandbox,--disable-setuid-sandbox,--disable-dev-shm-usage,--disable-accel
```

```
{
  "rendering": {
    "args": [
      "--no-sandbox",
      "--disable-setuid-sandbox",
      "--disable-dev-shm-usage",
      "--disable-accelerated-2d-canvas",
      "--disable-gpu",
      "--window-size=1280x758"
    ]
  }
}
```

Ignore HTTPS errors Instruct headless browser instance whether to ignore HTTPS errors during navigation. Per default HTTPS errors are not ignored. Due to the security risk it's not recommended to ignore HTTPS errors.

```
IGNORE_HTTPS_ERRORS=true
```

```
{
  "rendering": {
    "ignoresHttpsErrors": true
  }
}
```

Default timezone Instruct headless browser instance to use a default timezone when not provided by Grafana, .e.g. when rendering panel image of alert. See ICU's metaZones.txt for a list of supported timezone IDs. Fallbacks to `TZ` environment variable if not set.

```
BROWSER_TZ=Europe/Stockholm
```

```
{
  "rendering": {
    "timezone": "Europe/Stockholm"
  }
}
```

Default language Instruct headless browser instance to use a default language when not provided by Grafana, e.g. when rendering panel image of alert. Refer to the HTTP header Accept-Language to understand how to format this value.

```
{
  "rendering": {
    "acceptLanguage": "fr-CH, fr;q=0.9, en;q=0.8, de;q=0.7, *;q=0.5"
  }
}
```

Viewport width Default viewport width when width is not specified in the rendering request. Default is `1000`.

```
{
  "rendering": {
    "width": 1000
  }
}
```

Viewport height Default viewport height when height is not specified in the rendering request. Default is `500`.

```
{
  "rendering": {
    "height": 500
  }
}
```

Viewport maximum width Limit the maximum viewport width that can be requested. Default is `3000`.

```
{
  "rendering": {
    "maxWidth": 1000
  }
}
```

Viewport maximum height Limit the maximum viewport height that can be requested. Default is `3000`.

```
{
  "rendering": {
    "maxHeight": 500
  }
}
```

Device scale factor Specify default device scale factor for rendering images. `2` is enough for monitor resolutions, `4` would be better for printed material. Setting a higher value affects performance and memory. Default is `1`. This can be overridden in the rendering request.

```
{
  "rendering": {
    "deviceScaleFactor": 2
  }
}
```

Maximum device scale factor Limit the maximum device scale factor that can be requested. Default is `4`.

```
{
  "rendering": {
    "maxDeviceScaleFactor": 4
  }
}
```