+++ title = “Troubleshooting” description = “Image rendering troubleshooting” keywords = [“grafana”, “image”, “rendering”, “plugin”, “troubleshooting”] weight = 115 +++

Troubleshoot image rendering

In this section, you’ll learn how to enable logging for the image renderer and you’ll find the most common issues.

Enable debug logging

To troubleshoot the image renderer, different kind of logs are available.

You can enable debug log messages for rendering in the Grafana configuration file and inspect the Grafana server logs.

```
[log]
filters = rendering:debug
```

You can also enable more logs in image renderer service itself by:

- Increasing the [log level]({{< relref “./#log-level” >}}).
- Enabling [verbose logging]({{< relref “./#verbose-logging” >}}).
- [Capturing headless browser output]({{< relref “./#capture-browser-output” >}}).

Missing libraries

The plugin and rendering service uses Chromium browser which depends on certain libraries. If you don’t have all of those libraries installed in your system you may encounter errors when trying to render an image, e.g.

```
Rendering failed: Error: Failed to launch chrome!/var/lib/grafana/plugins/grafana-image-renc
error while loading shared libraries: libX11.so.6: cannot open shared object file: No such f
```

In general you can use the `ldd` utility to figure out what shared libraries are not installed in your system:

```
cd <grafana-image-render plugin directory>
ldd chrome-linux/chrome
      linux-vdso.so.1 (0x00007fff1bf65000)
      libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f2047945000)
      libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f2047924000)
      librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1 (0x00007f204791a000)
      libX11.so.6 => not found
      libX11-xcb.so.1 => not found
      libxcb.so.1 => not found
      libXcomposite.so.1 => not found
        ...
```

Ubuntu:

On Ubuntu 18.10 the following dependencies are required for the image rendering to function.

```
libx11-6 libx11-xcb1 libxcomposite1 libxcursor1 libxdamage1 libxext6 libxfixes3 libxi6 libxr
```

Debian:

On Debian 9 (Stretch) the following dependencies are required for the image rendering to function.

```
libx11 libcairo libcairo2 libxtst6 libxcomposite1 libx11-xcb1 libxcursor1 libxdamage1 libnss
```

On Debian 10 (Buster) the following dependencies are required for the image rendering to function.

```
libxdamage1 libxext6 libxi6 libxtst6 libnss3 libcups2 libxss1 libxrandr2 libasound2 libatk1.
```

Centos:

On a minimal CentOS 7 installation, the following dependencies are required for the image rendering to function:

```
libXcomposite libXdamage libXtst cups libXScrnSaver pango atk adwaita-cursor-theme adwaita-i
```

On a minimal CentOS 8 installation, the following dependencies are required for the image rendering to function:

```
libXcomposite libXdamage libXtst cups libXScrnSaver pango atk adwaita-cursor-theme adwaita-i
```

Certificate signed by internal certificate authorities

In many cases, Grafana runs on internal servers and uses certificates that have not been signed by a CA (Certificate Authority) known to Chrome, and therefore cannot be validated. Chrome internally uses NSS (Network Security Services) for cryptographic operations such as the validation of certificates.

If you are using the Grafana Image Renderer with a Grafana server that uses a certificate signed by such a custom CA (for example a company-internal CA), rendering images will fail and you will see messages like this in the Grafana log:

```
t=2019-12-04T12:39:22+0000 lvl=error msg="Render request failed" logger=rendering error=map
t=2019-12-04T12:39:22+0000 lvl=error msg="Rendering failed." logger=context userId=1 orgId=1
t=2019-12-04T12:39:22+0000 lvl=error msg="Request Completed" logger=context userId=1 orgId=1
```

If this happens, then you have to add the certificate to the trust store. If you have the certificate file for the internal root CA in the file `internal-root-ca.crt.pem`, then use these commands to create a user specific NSS trust store for the Grafana user (`grafana` for the purpose of this example) and execute the following steps:

Linux:

```
[root@server ~]# [ -d /usr/share/grafana/.pki/nssdb ] || mkdir -p /usr/share/grafana/.pki/ns
[root@merver ~]# certutil -d sql:/usr/share/grafana/.pki/nssdb -A -n internal-root-ca -t C -
[root@server ~]# chown -R grafana: /usr/share/grafana/.pki/nssdb
```

Windows:

```
certutil -addstore "Root" <path>/internal-root-ca.crt.pem
```

Custom Chrome/Chromium

As a last resort, if you already have Chrome or Chromium installed on your system, then you can configure the Grafana Image renderer plugin to 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.

To override the path to the Chrome/Chromium executable in plugin mode, set an environment variable and make sure that it's available for the Grafana process. For example:

```
export GF_PLUGIN_RENDERING_CHROME_BIN="/usr/bin/chromium-browser"
```

In remote rendering mode, you need to set the environment variable or update the configuration file and make sure that it's available for the image rendering service process:

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

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