SOFTWARE
ARCHITECTURE

Escuela de Ingeniería Informática

Universidad de Oviedo

Lab 11

Monitoring and profiling: observability

2023-24

Jose Emilio Labra Gayo
Pablo González
Cristian Augusto Alonso
Jorge Álvarez Fidalgo

Monitoring and profiling

Monitoring: Observe the behaviour at runtime while software is running
- Dashboards
- Usually, after deployment

Profiling: Measure performance of a software while it is running
- Identify parts of a system that contribute to a performance problem
- Show where to concentrate the efforts
- Usually before deployment

Monitoring & profiling

Monitors an application while it is running

Records performance (CPU & memory usage)

JavaScript:

- Chrome (Timeline), Firefox Developer Edition (Performance tool)

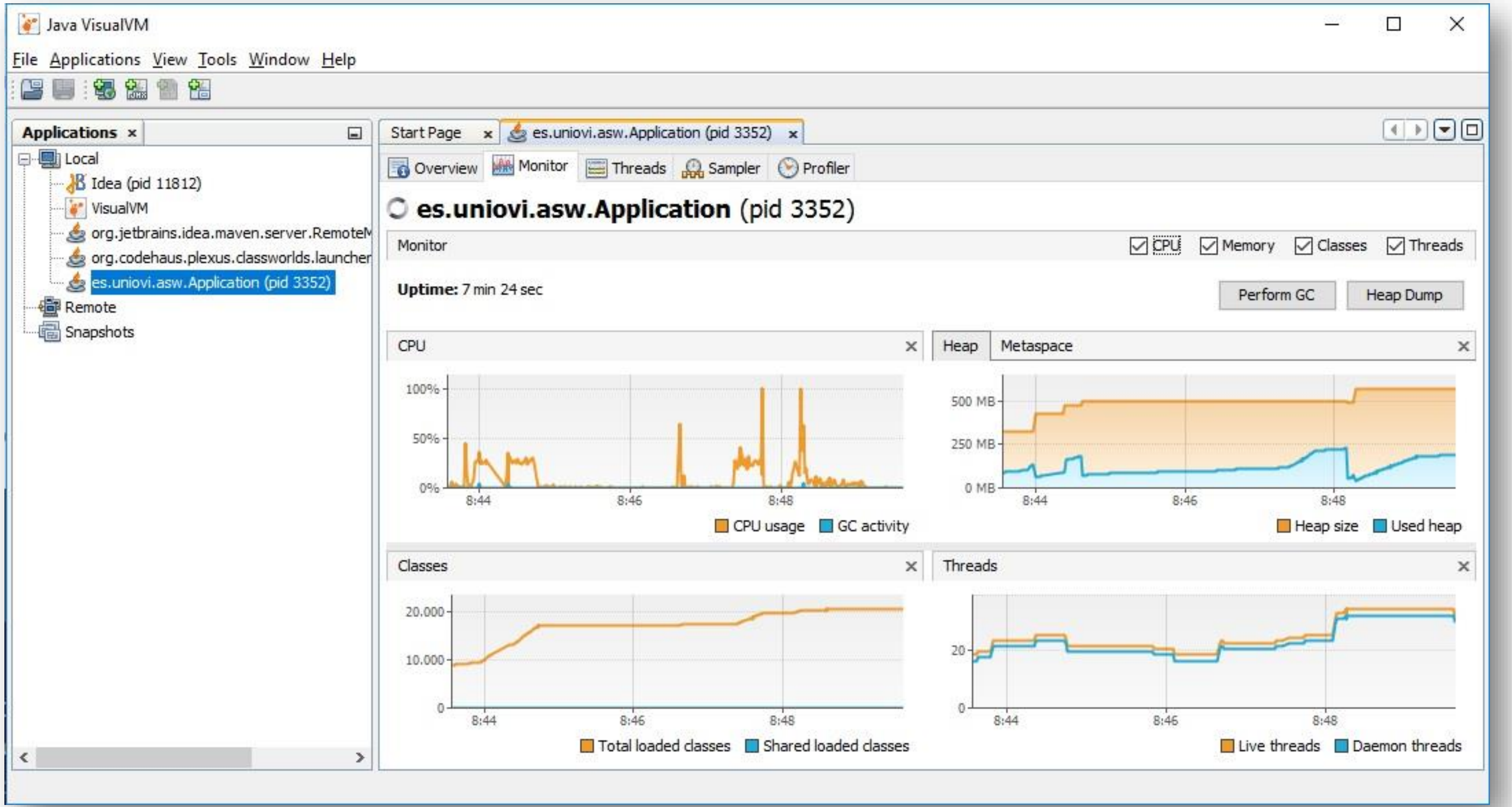
Server-side:

- JVisualVM, JProfiler, YourKit, JConsole
- Monitoring: Graphite, Datadog, Prometheus, Graphana

VisualVM

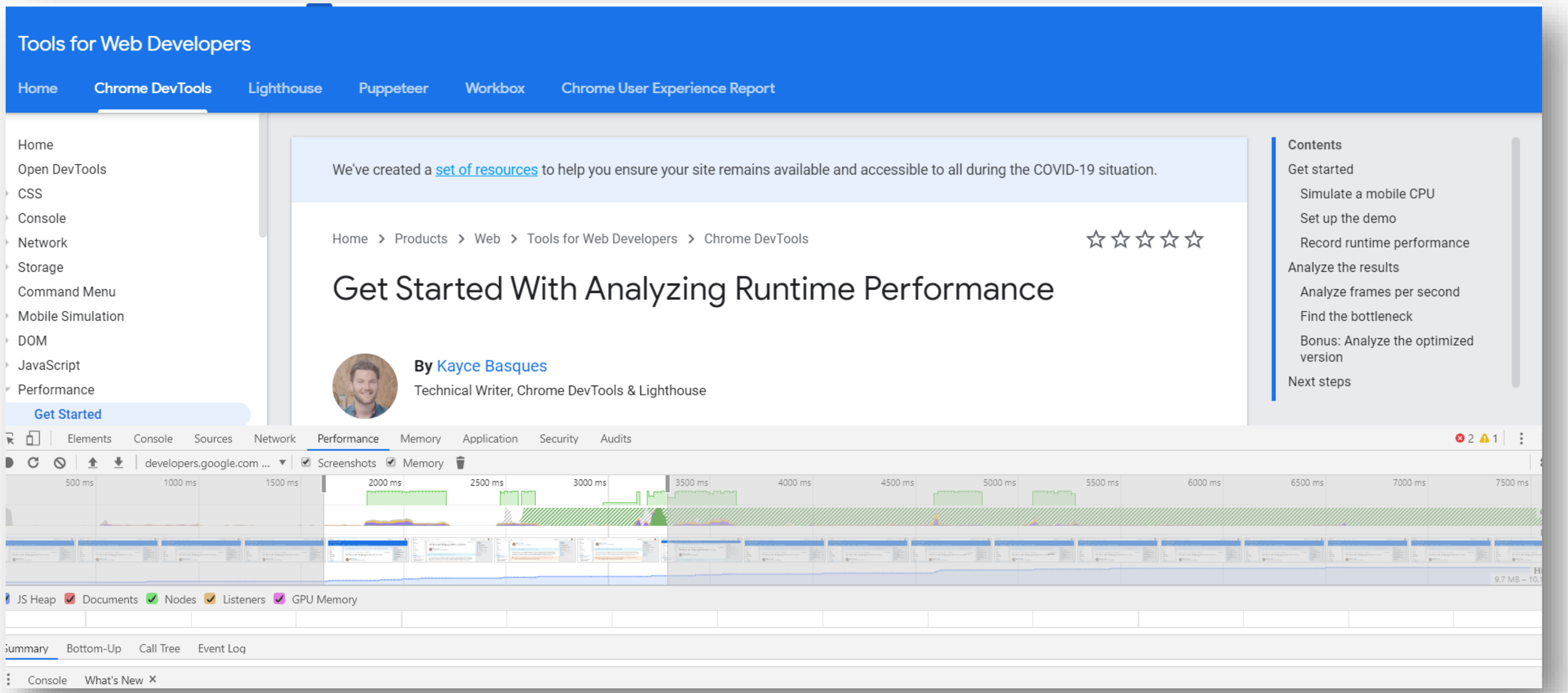
- https://visualvm.github.io/
- `jvisualvm`

Java/server JVisualVM



Browser: developer tools

Profiling/check performance



We've created a set of resources to help you ensure your site remains available and accessible to all during the COVID-19 situation.

Home > Products > Web > Tools for Web Developers > Chrome DevTools

Get Started With Analyzing Runtime Performance

By Kayce Basques
Technical Writer, Chrome DevTools & Lighthouse

https://developers.google.com/web/tools/chrome-devtools/evaluate-performance

Example with Google Chrome

Incognito mode

At the top right, click the three dots and then New Incognito Window.

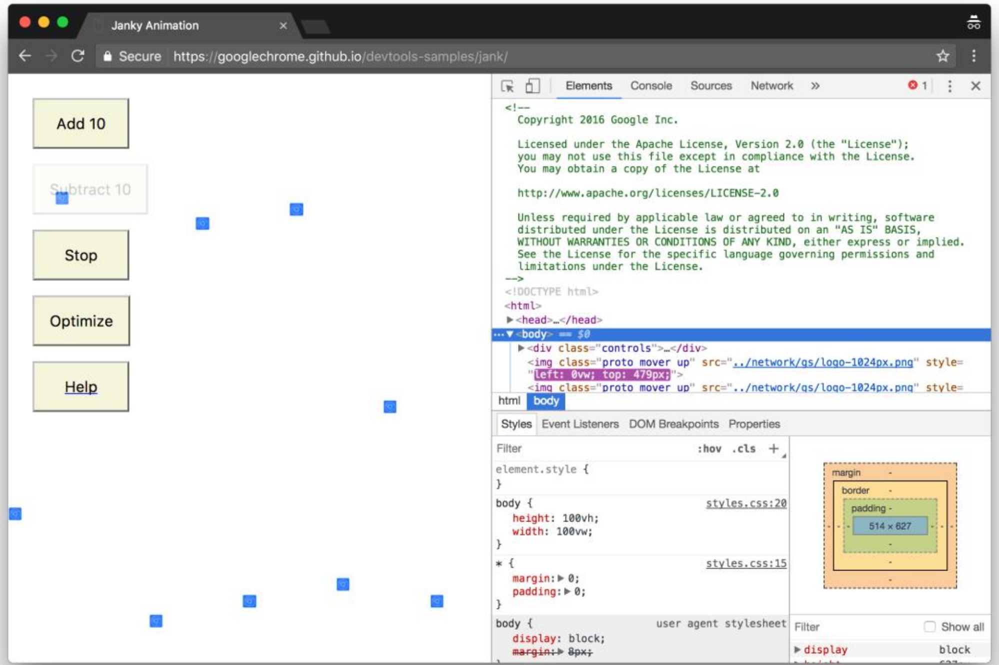
Windows, Linux, or Chrome OS: Press Ctrl + Shift + n.
Mac: Press ⌘ + Shift + n.

DevTools

Windows, Linux: Control+Shift+I
Mac: Command+Option+I

Example with Google Chrome

https://googlechrome.github.io/devtools-samples/jank/



```
<!--
  Copyright 2016 Google Inc.

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body> == $0
  ▶<div class="controls">…</div>
   <img class="proto mover up" src="../network/gs/logo-1024px.png" style=
   "left: 0vw; top: 479px;">
   <img class="proto mover up" src="../network/gs/logo-1024px.png" style=
```

```
element.style {
}
body {                                    styles.css:20
  height: 100vh;
  width: 100vw;
}
* {                                       styles.css:15
  margin:▶ 0;
  padding:▶ 0;
}
body {                           user agent stylesheet
  display: block;
  ~~margin:▶ 8px;~~
```

```
▶ display                                  block
```

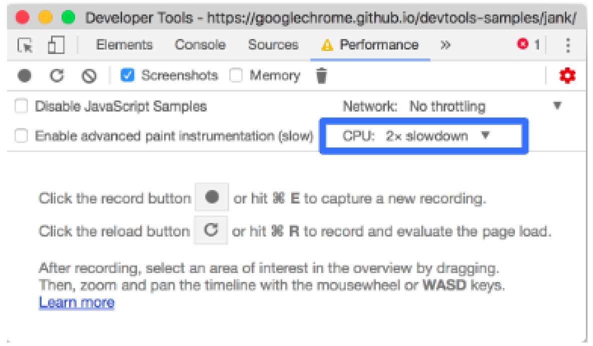
Performance>Record

click Add 10 (20 times)

try Optimize / Un-optimize

Stop

Performance>CPU>2 x Slowdown


Developer Tools - https://googlechrome.github.io/devtools-samples/jank/
Elements
Console
Sources
Performance
Screenshots
Memory

Disable JavaScript Samples — Network: No throttling

Enable advanced paint instrumentation (slow) — CPU: 2× slowdown

Click the record button ● or hit ⌘ E to capture a new recording.

Click the reload button ⟳ or hit ⌘ R to record and evaluate the page load.

After recording, select an area of interest in the overview by dragging. Then, zoom and pan the timeline with the mousewheel or WASD keys. Learn more

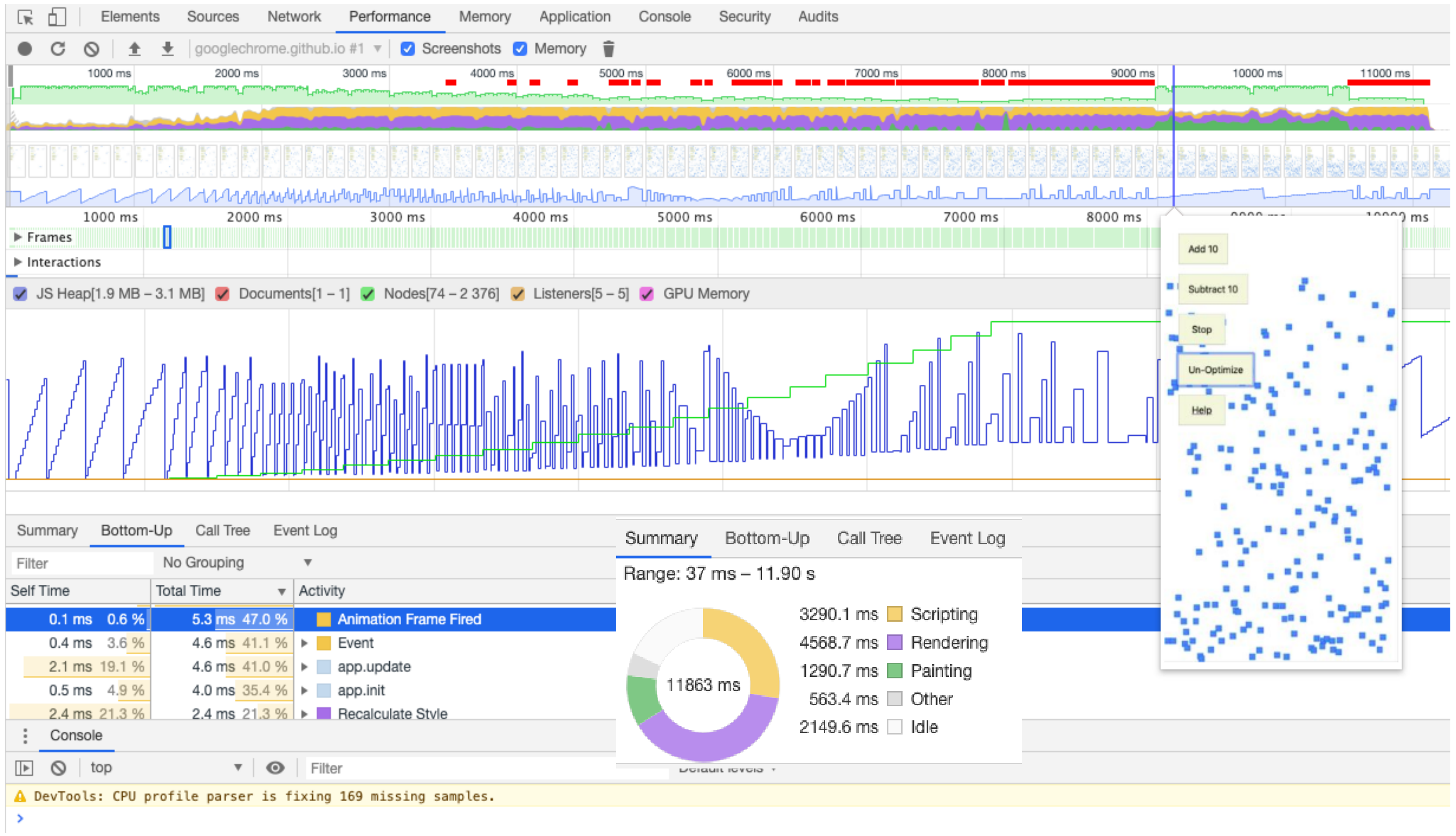
Example with Google Chrome

Profile result:

Frames per Second

CPU

Bottleneck



Self Time		Total Time		Activity
0.1 ms	0.6 %	5.3 ms	47.0 %	🟨 Animation Frame Fired
0.4 ms	3.6 %	4.6 ms	41.1 %	▶ 🟨 Event
2.1 ms	19.1 %	4.6 ms	41.0 %	▶ 🟦 app.update
0.5 ms	4.9 %	4.0 ms	35.4 %	▶ 🟦 app.init
2.4 ms	21.3 %	2.4 ms	21.3 %	▶ 🟪 Recalculate Style

Range: 37 ms – 11.90 s

11863 ms

3290.1 ms	🟨	Scripting
4568.7 ms	🟪	Rendering
1290.7 ms	🟩	Painting
563.4 ms	⬜	Other
2149.6 ms	⬜	Idle

```
⚠ DevTools: CPU profile parser is fixing 169 missing samples.
```

Other tools for browser

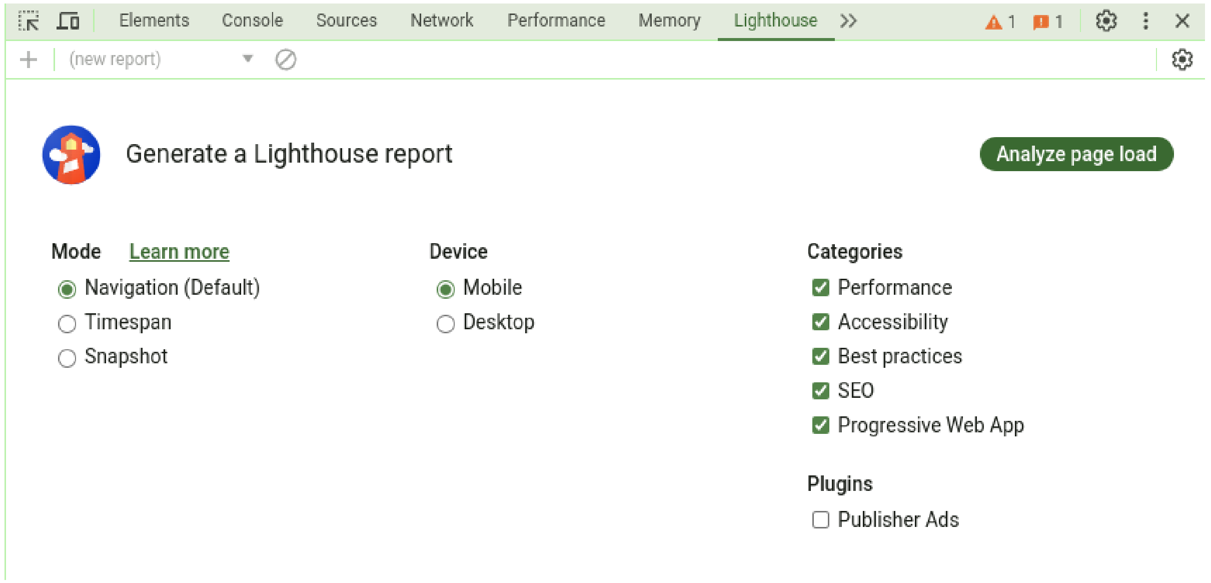
RAIL model:

Response, Animation, Idle, Load

https://developers.google.com/web/fundamentals/performance/rail

https://webpagetest.org/easy

Lighthouse (with Chrome)



Generate a Lighthouse report

Mode
- [✓ Navigation (Default)]
- Timespan
- Snapshot

Device
- [✓ Mobile]
- Desktop

Categories
- [✓ Performance]
- [✓ Accessibility]
- [✓ Best practices]
- [✓ SEO]
- [✓ Progressive Web App]

Plugins
- Publisher Ads

Server side monitoring

- Cloud platforms like Azure provide monitoring solutions
    - Also available in Google Cloud, Amazon AWS, Alibaba Cloud…
    - In the case of Azure: Azure Monitor
- There is also the option to set up our own monitoring solution
- Which software to use: *Prometheus* and *Graphana*
- Guide: https://github.com/Arquisoft/wiq_0/blob/master/gatewayservice/README.mdd

Server side monitoring

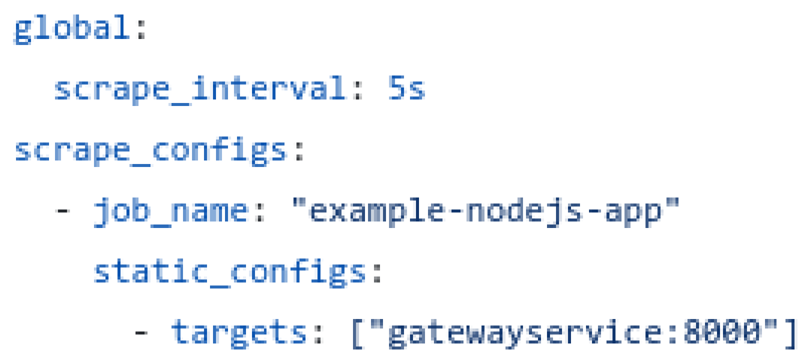
- We need a library that can extract some metrics from our gatewayservice
    - *npm install prom-client express-prom-bundle*

```
const metricsMiddleware:RequestHandler = promBundle({includeMethod: true});
app.use(metricsMiddleware);
```

- If we launch the gatewayservice, in */metrics* we will be able to see some row data that would be used by Graphana to plot nice charts
- We can choose which metrics to measure [doc]

Server side monitoring

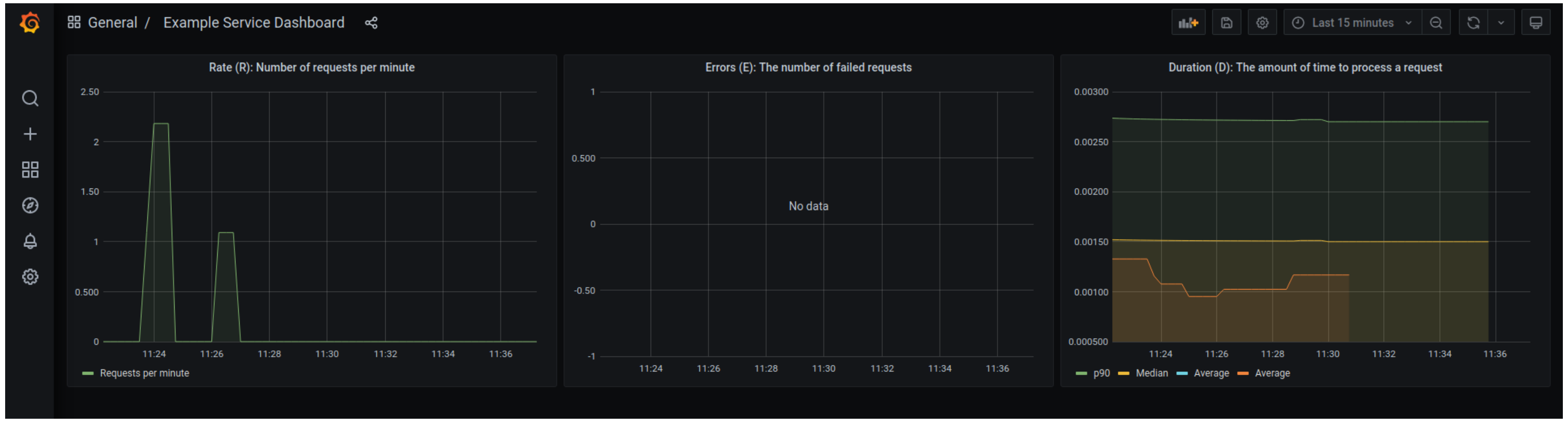
- Graphana cannot use this data directly, we need Prometheus
  - Prometheus will retrieve the data exposed by the service (e.g. gateway) and store it so it can be consumed by Graphana
  - We will work with a docker image [prom/prometheus] that can be configured through a single file



```
global:
  scrape_interval: 5s
scrape_configs:
  - job_name: "example-nodejs-app"
    static_configs:
      - targets: ["gatewayservice:8000"]
```

Server side monitoring

- How to configure Graphana
  - Graphana will use Prometheus as data source
  - We also have a docker image for running it [grafana/grafana]
  - We need to configure the datasource and the dashboard (which charts to plot)


General / Example Service Dashboard
Last 15 minutes
Rate (R): Number of requests per minute
Requests per minute
Errors (E): The number of failed requests
No data
Duration (D): The amount of time to process a request
p90
Median
Average
Average

Links

Monitoring & Profiling

Get Started With Analyzing Runtime Performance
https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/

How to Use the Timeline Tool
https://developers.google.com/web/tools/chrome-devtools/evaluate-performance timeline-tool#profile-js