



ARQUITECTURA DEL SOFTWARE

2023-24

Jose Emilio Labra Gayo

Pablo González

Cristian Augusto Alonso

Jorge Álvarez Fidalgo



Escuela de
Ingeniería
Informática



Universidad de Oviedo

Laboratorio 12

Monitorización
Observabilidad

Monitorización y Profiling

- **Monitorizar:** Observar comportamiento de un software
 - Cuadros de mando
 - Habitualmente, después del despliegue
- **Profiling (caracterizar):** Medir rendimiento de un software mientras se ejecuta
 - Identificar partes que contribuyen a un problema
 - Mostrar dónde centrar los esfuerzos para mejorar rendimiento
 - Suele hacerse antes del despliegue

Monitorización y Profiling

Monitorizar una aplicación mientras se ejecuta
Registrar uso de CPU, memoria, hilos, etc.

JavaScript:

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

Herramientas de servidor:

JVisualVM, JProfiler, YourKit, Jconsole, etc.

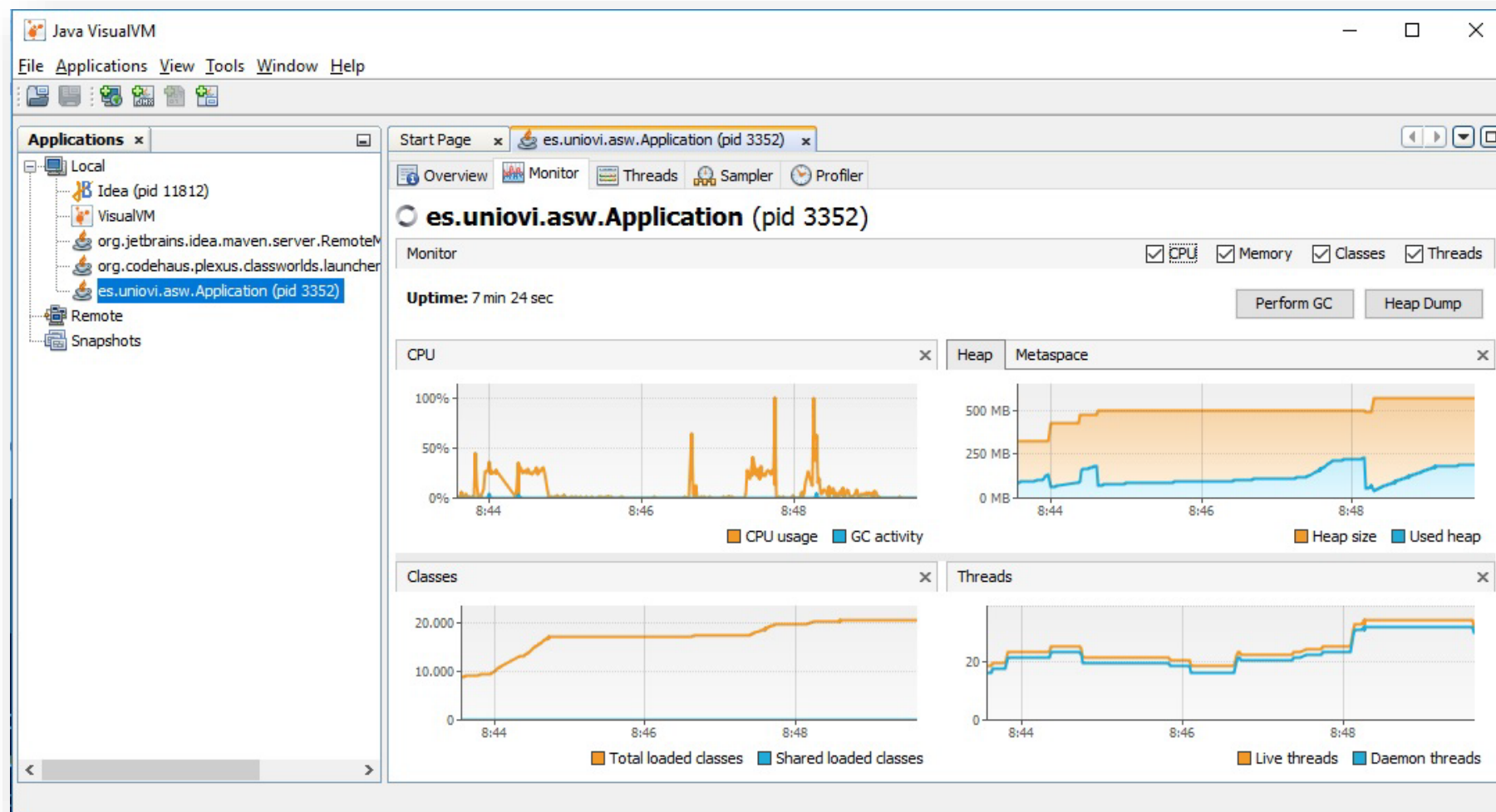
Graphite, Datdog, Prometheus, Graphana

VisualVM

<https://visualvm.github.io/>

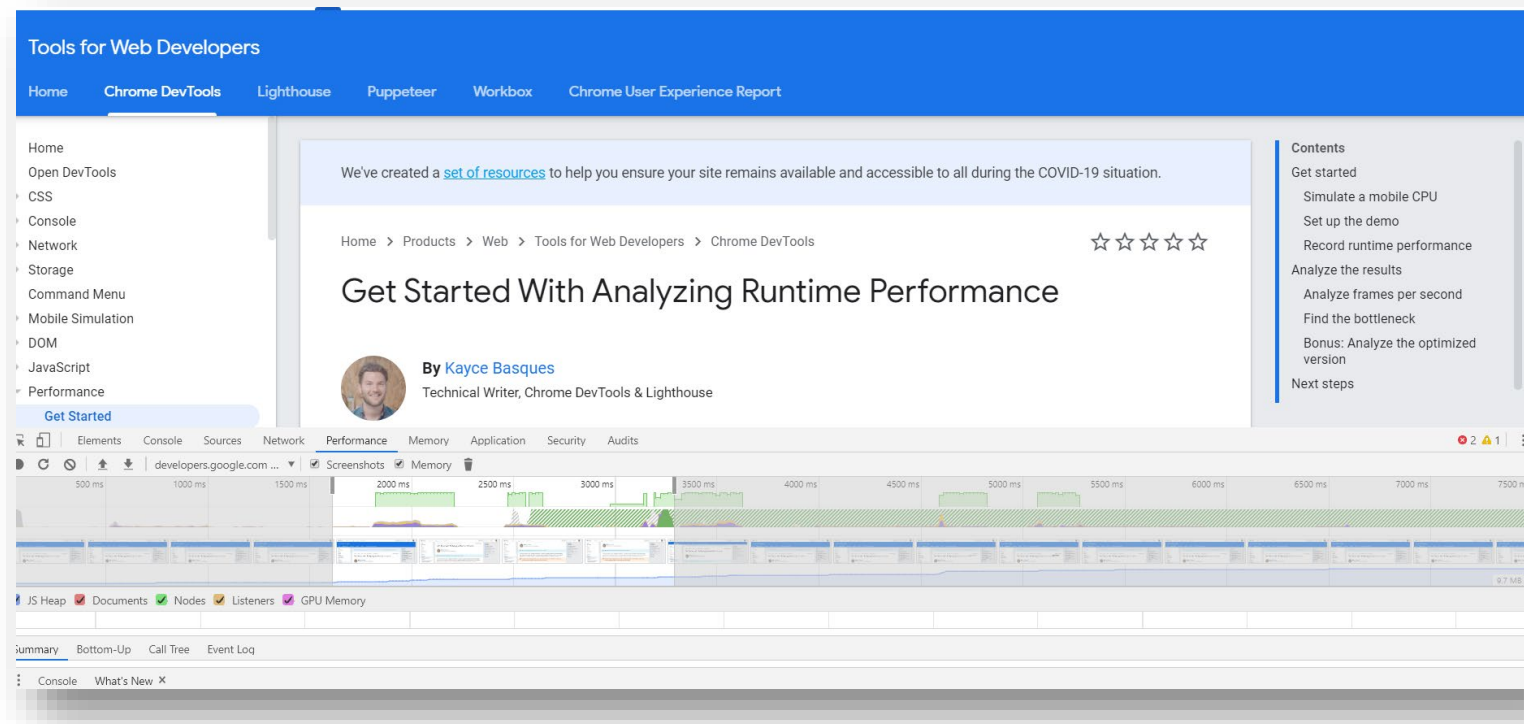
Ya está instalada con el JDK: `jvisualvm`

Server/Java: JVisualVM



Navegador: developer tools

- Monitorizar/chequear rendimiento



The screenshot displays the 'Tools for Web Developers' interface, specifically the Performance tab. The main content area features a 'Get Started With Analyzing Runtime Performance' article by Kayce Basques, a Technical Writer at Chrome DevTools & Lighthouse. The article includes a navigation breadcrumb: 'Home > Products > Web > Tools for Web Developers > Chrome DevTools' and a five-star rating. A 'Contents' sidebar on the right lists sections such as 'Get started', 'Simulate a mobile CPU', 'Set up the demo', 'Record runtime performance', 'Analyze the results', 'Analyze frames per second', 'Find the bottleneck', 'Bonus: Analyze the optimized version', and 'Next steps'. The bottom of the screenshot shows the Performance timeline with a memory graph and various tool panels like 'JS Heap', 'Documents', 'Nodes', 'Listeners', and 'GPU Memory'.

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

Ejemplo: Google Chrome

Modo incognito

En la esquina superior derecha, click en los tres puntos y nueva ventana incógnito
Windows, Linux, or Chrome OS: Ctrl + Shift + n.

Mac: ⌘ + Shift + n.

Chrome DevTools

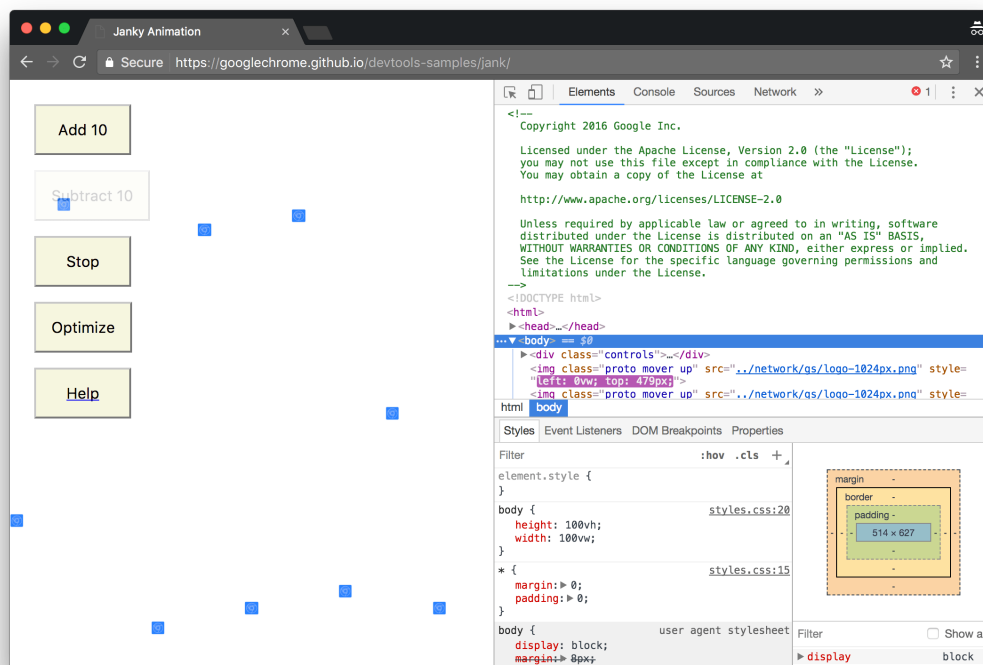
Windows, Linux: Control+Shift+I

Mac: Command+Option+I

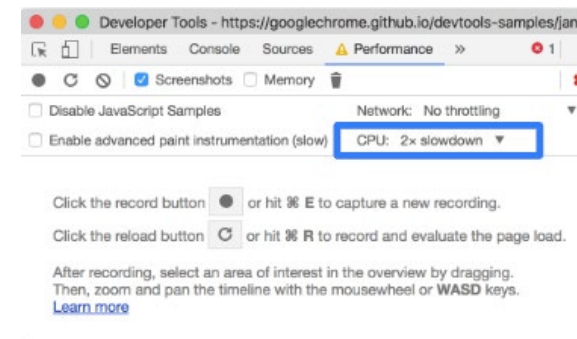


Ejemplo: Google Chrome

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



Performance > CPU > 2 x Slowdown



Performance > Record

click Add 10 (20 veces)

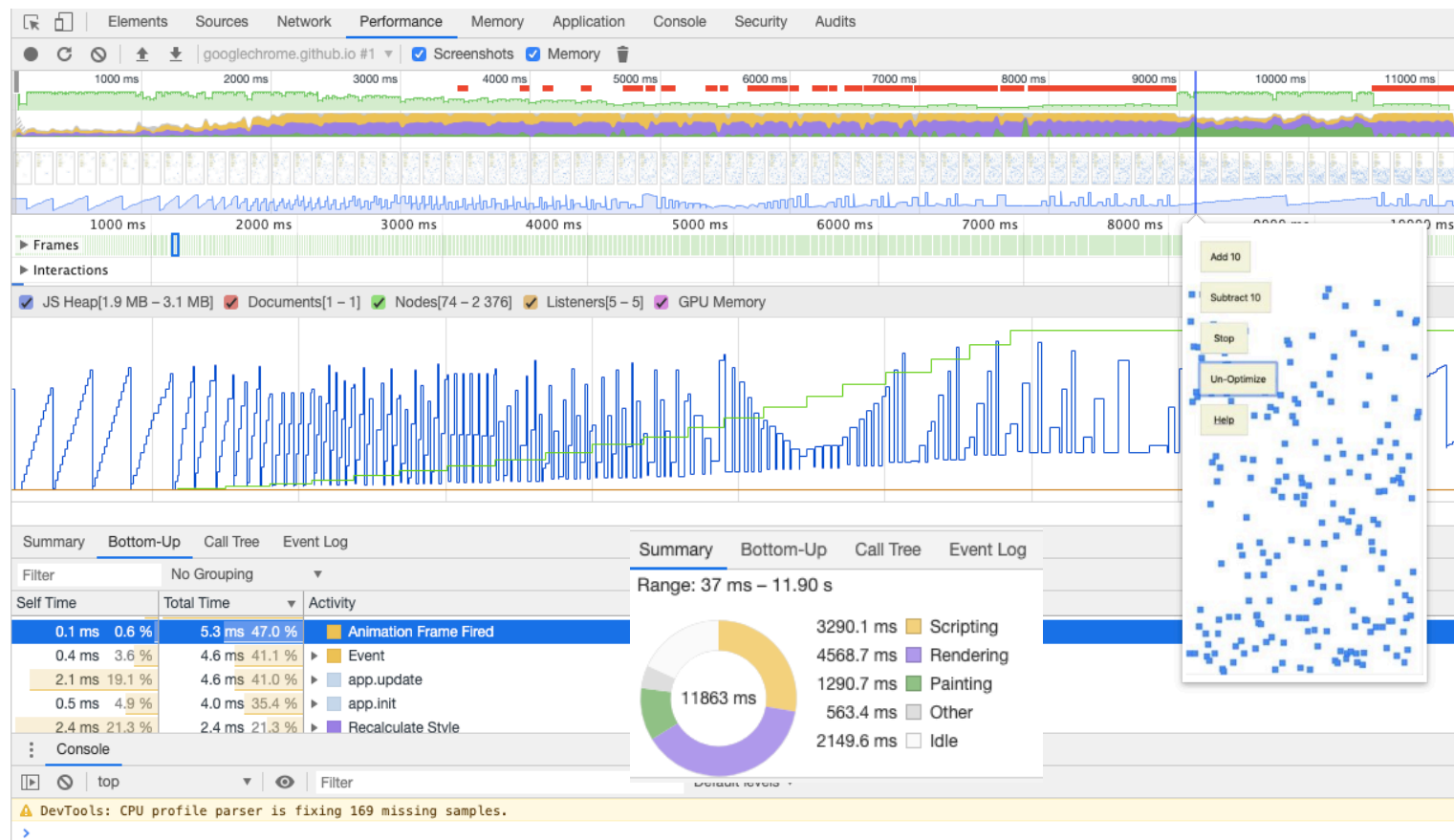
Optimize / Un-optimize

Stop

Ejemplo: Google Chrome

Profile result:

Frames por segundo →
CPU →



Cuellos de botella →

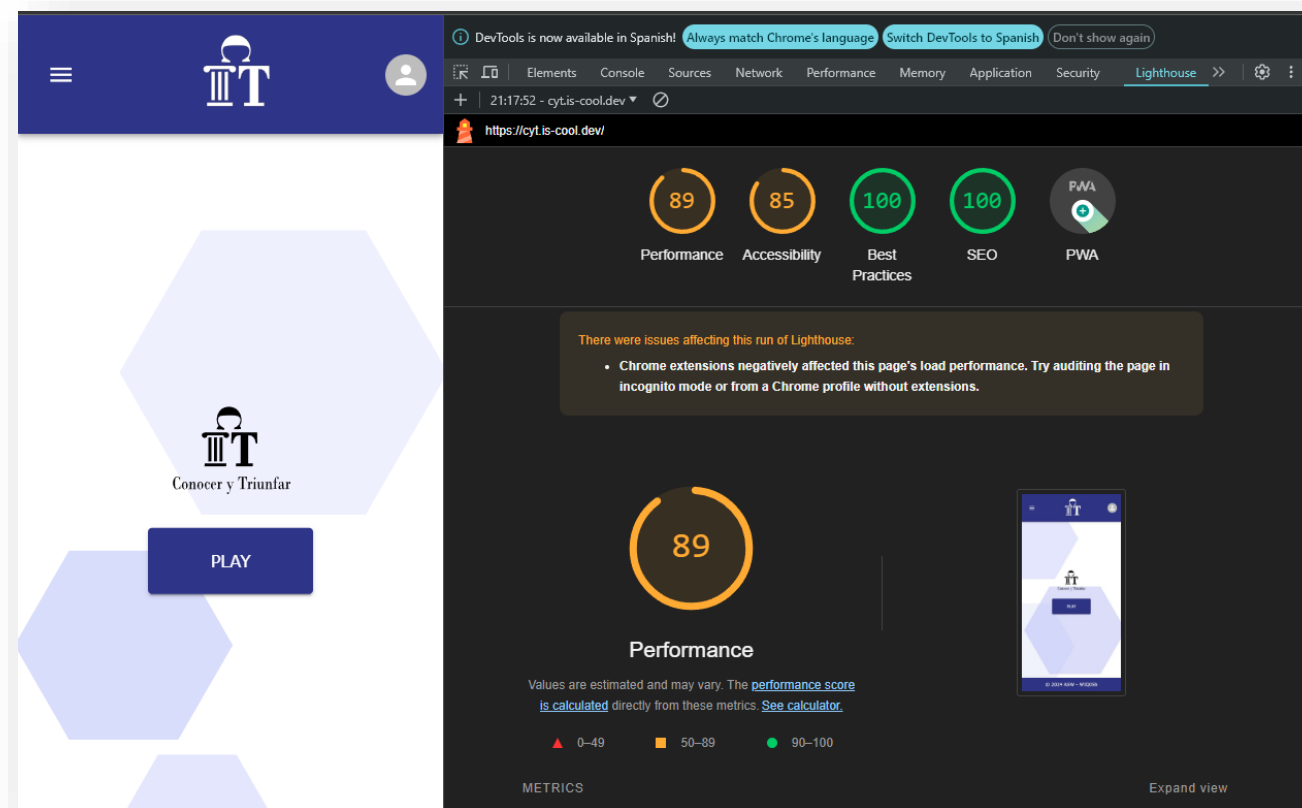
Otras herramientas de navegador

RAIL model (Response,
Animation, Idle, Load)

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

<https://webpagetest.org/easy>

Lighthouse (incluido en Chrome)



Monitorización en servidor

Las plataformas en la nube brindan soluciones de monitoreo

También disponible en Google Cloud, Amazon AWS, Alibaba Cloud...

En el caso de Azure: [Azure Monitor](#)

Aunque también existen soluciones de terceros

Prometheus, Graphite, Grafana, Datadog, Nagios, Sensu, ...

Usaremos: **Prometheus** y **Grafana**

wiq_0:

https://github.com/arquisoft/wiq_0/tree/master/gatewayservice#monitoring-prometheus-and-grafana



- **Prometheus:** servidor de almacenamiento de datos en series de tiempo
 - Modelo de datos multidimensional
 - Lenguaje flexible de consultas
 - Nodos autónomos de servidor único
 - Configuración estática
- **Grafana:** Visualización de datos. Permite crear, explorar y compartir tableros

Monitorización en servidor

- Necesitamos una biblioteca que pueda extraer algunas métricas de nuestro servicio (e.g. Gatewayservice)

1. Instalar el cliente

```
npm install prom-client express-prom-bundle
```

2. Modificamos *gatewayservice/gateway-service.js*

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

3. Si lanzamos el *gatewayservice*, en */metrics* podremos ver algunos datos de fila que Graphana usaría para trazar los gráficos.
Podemos elegir qué métrica medir [[doc](#)]



Monitorización en servidor

- Graphana no puede usar esta información directamente, necesita Prometheus
 - Prometheus recuperará los datos expuestos por el servicio (e.g. gateway-service) y los almacenará para que Grafana pueda consumirlos.
 - Trabajaremos con una docker image [prom/prometheus] que se puede configurar a través de un solo archivo

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



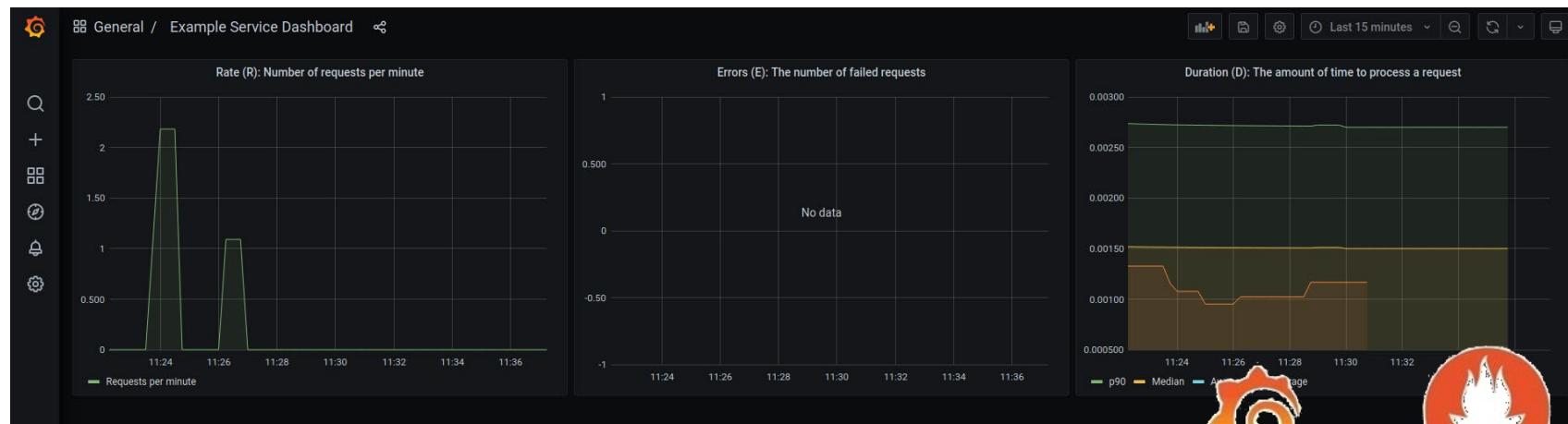
Grafana



Prometheus

Monitorización en servidor

- Como configurar Graphana
 - Graphana usará Prometheus como fuente de datos
 - Tenemos una docker image para ejecutarlo [grafana/grafana]
 - Nosotros necesitamos configurar datasource y el dashboard (gráficos a visualizar)



Referencias

- Monitorización y Profiling
 - [Get Started With Analyzing Runtime Performance](https://developers.google.com/web/tools/chrome-devtools/evaluate-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)
<https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/timeline-tool#profile-js>
 - [Otro Ejemplo](https://github.com/coder-society/nodejs-application-monitoring-with-prometheus-and-grafana)
<https://github.com/coder-society/nodejs-application-monitoring-with-prometheus-and-grafana>