



**SOFTWARE
ARCHITECTURE**

2025-26

Jose Emilio Labra Gayo

Pablo González

Diego Martín

Celia Melendi



Escuela de
Ingeniería
Informática



Universidad de Oviedo

Lab 11

Monitoring and profiling: observability

Monitoring and profiling

Quality attribute: Observability

Monitoring: Observe the behaviour at runtime while software is running

Dashboards

Usually in production (after deployment)

Profiling: Measure performance of a software while it is running

Identify parts of a system that contribute to performance problems

Find where to focus the efforts to improve performance

Usually when developing/testing (before deployment)

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

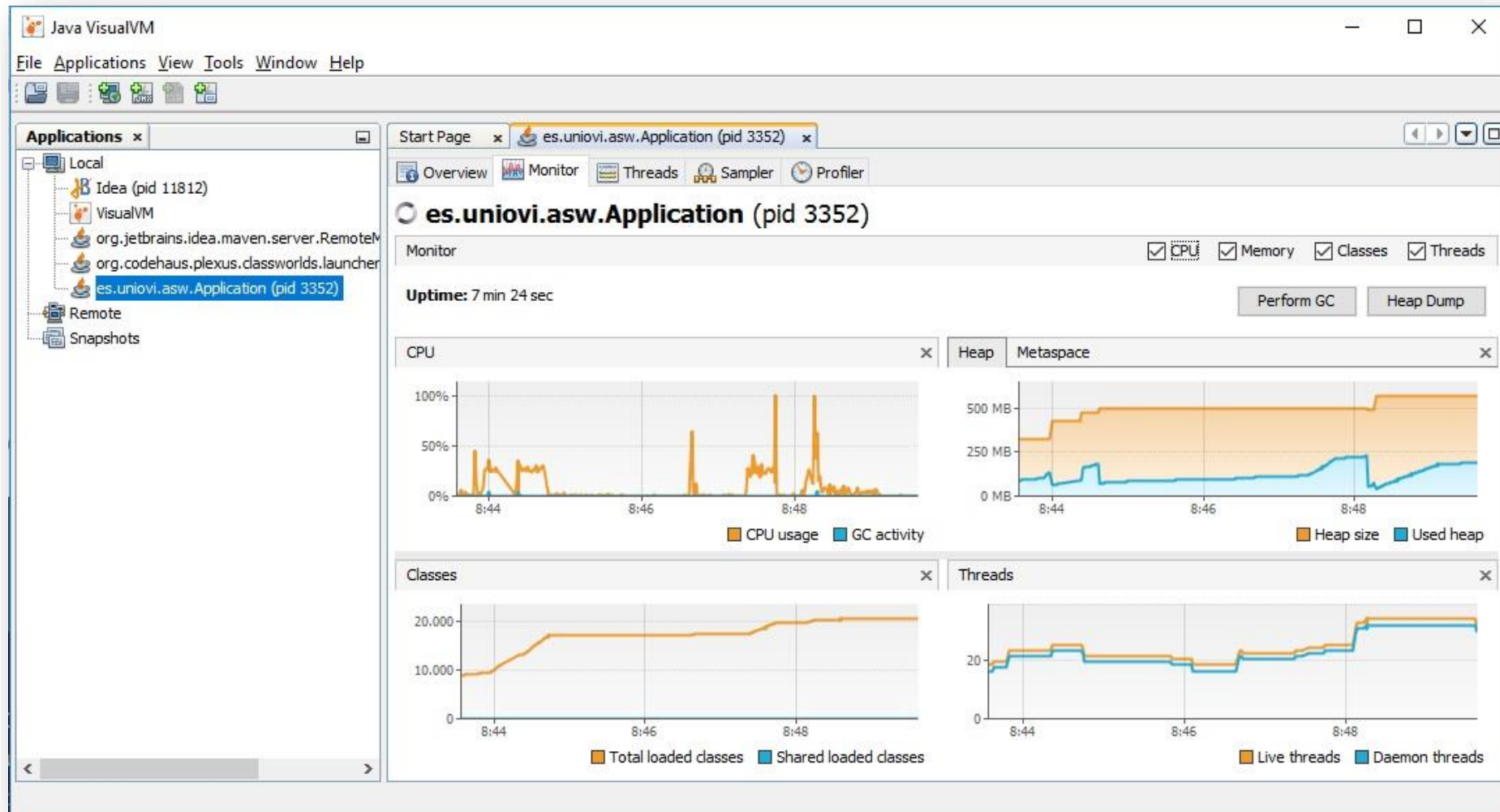
Monitoring: Graphite, Datadog, Prometheus, Graphana

VisualVM

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

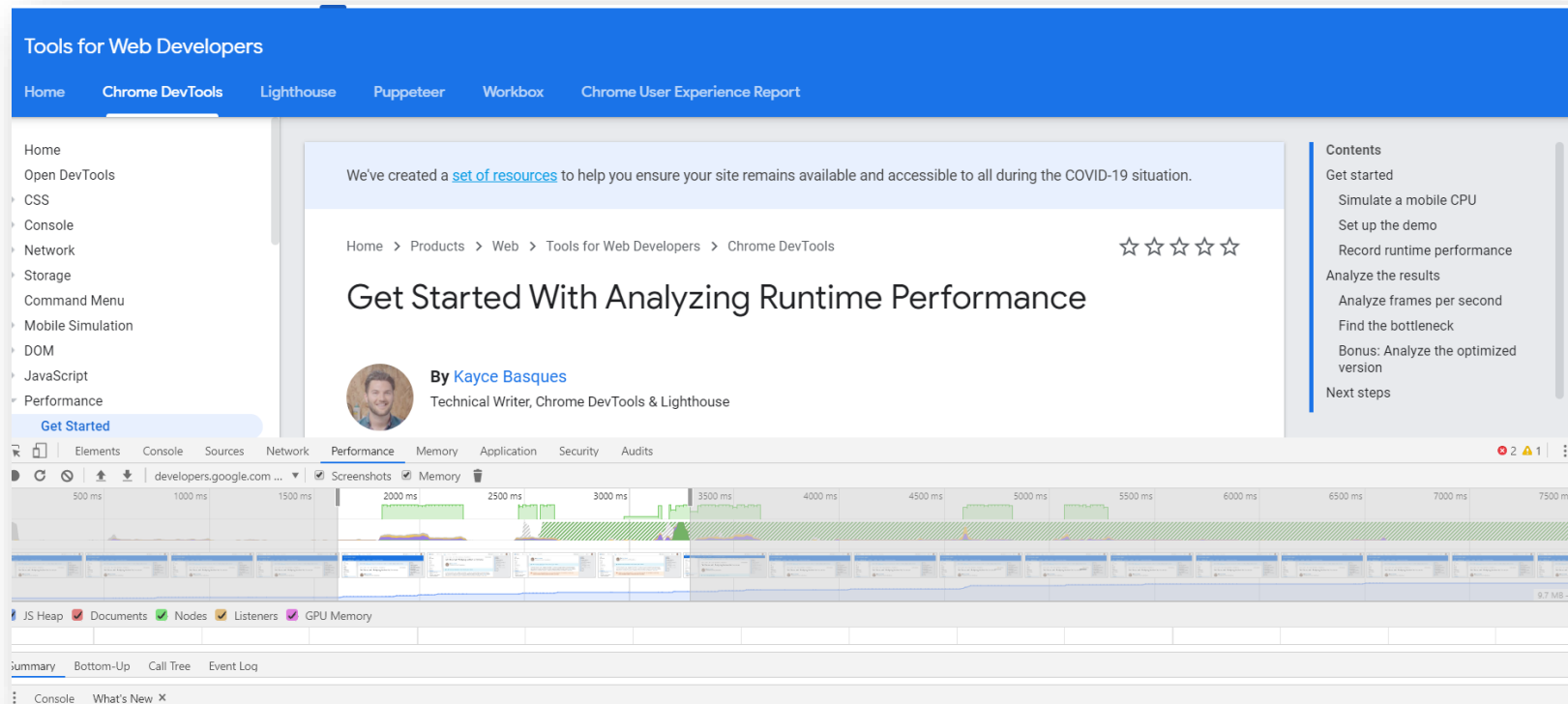
`jvisualvm`

Java/server JVisualVM



Browser: developer tools

Profiling/check performance



The screenshot displays the Chrome DevTools Performance page. The top navigation bar includes links for Home, Chrome DevTools, Lighthouse, Puppeteer, Workbox, and Chrome User Experience Report. The left sidebar lists various tool categories such as Home, Open DevTools, CSS, Console, Network, Storage, Command Menu, Mobile Simulation, DOM, JavaScript, and Performance. The main content area features a blue header with the text "Tools for Web Developers" and a navigation menu. Below this, there is a message about resources for COVID-19, a breadcrumb trail, and a five-star rating. The main heading is "Get Started With Analyzing Runtime Performance" by Kayce Basques, a Technical Writer for Chrome DevTools & Lighthouse. A right-hand sidebar contains a "Contents" section with links to "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 portion of the image shows the Performance tab interface, including a timeline with a green bar chart representing CPU usage, a list of tasks, and a legend for JS Heap, Documents, Nodes, Listeners, and GPU Memory. The bottom status bar shows "summary", "Bottom-Up", "Call Tree", "Event Log", "Console", and "What's New".

<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

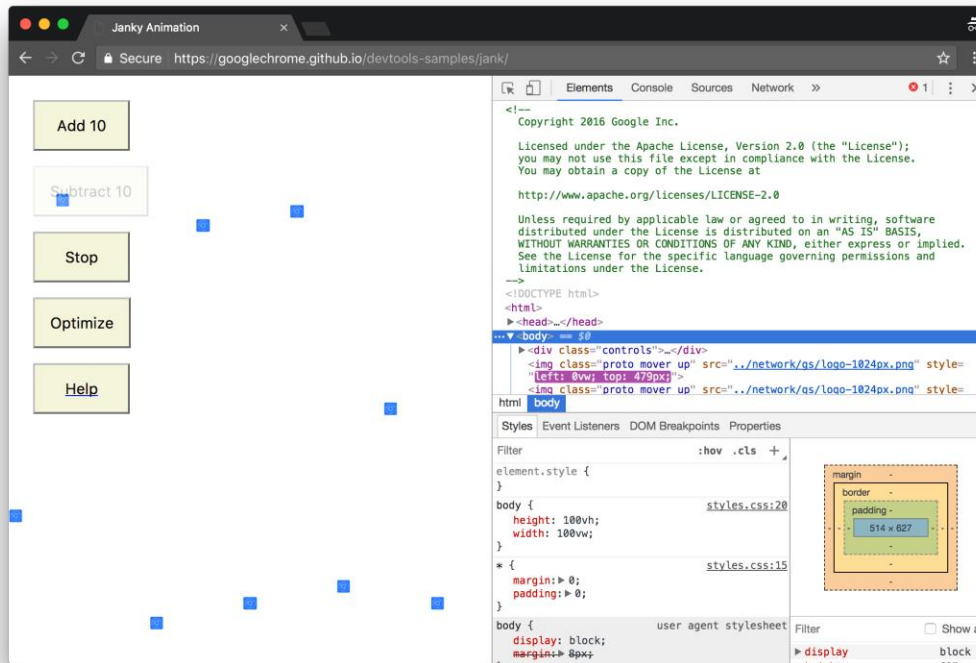
Windows, Linux: Control+Shift+I

Mac: Command+Option+I



Example with Google Chrome

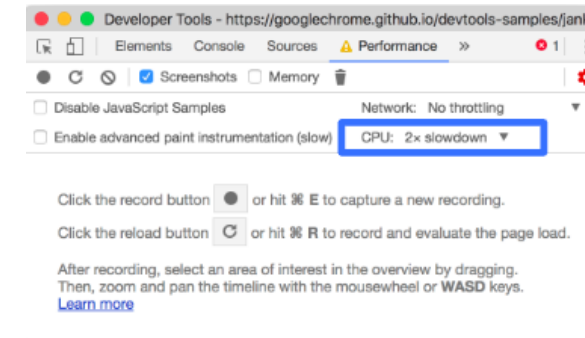
<https://googlechrome.github.io/devtools-samples/jank/>



Performance > Record
 click Add 10 (20 times)
 try Optimize / Un-optimize
 Stop



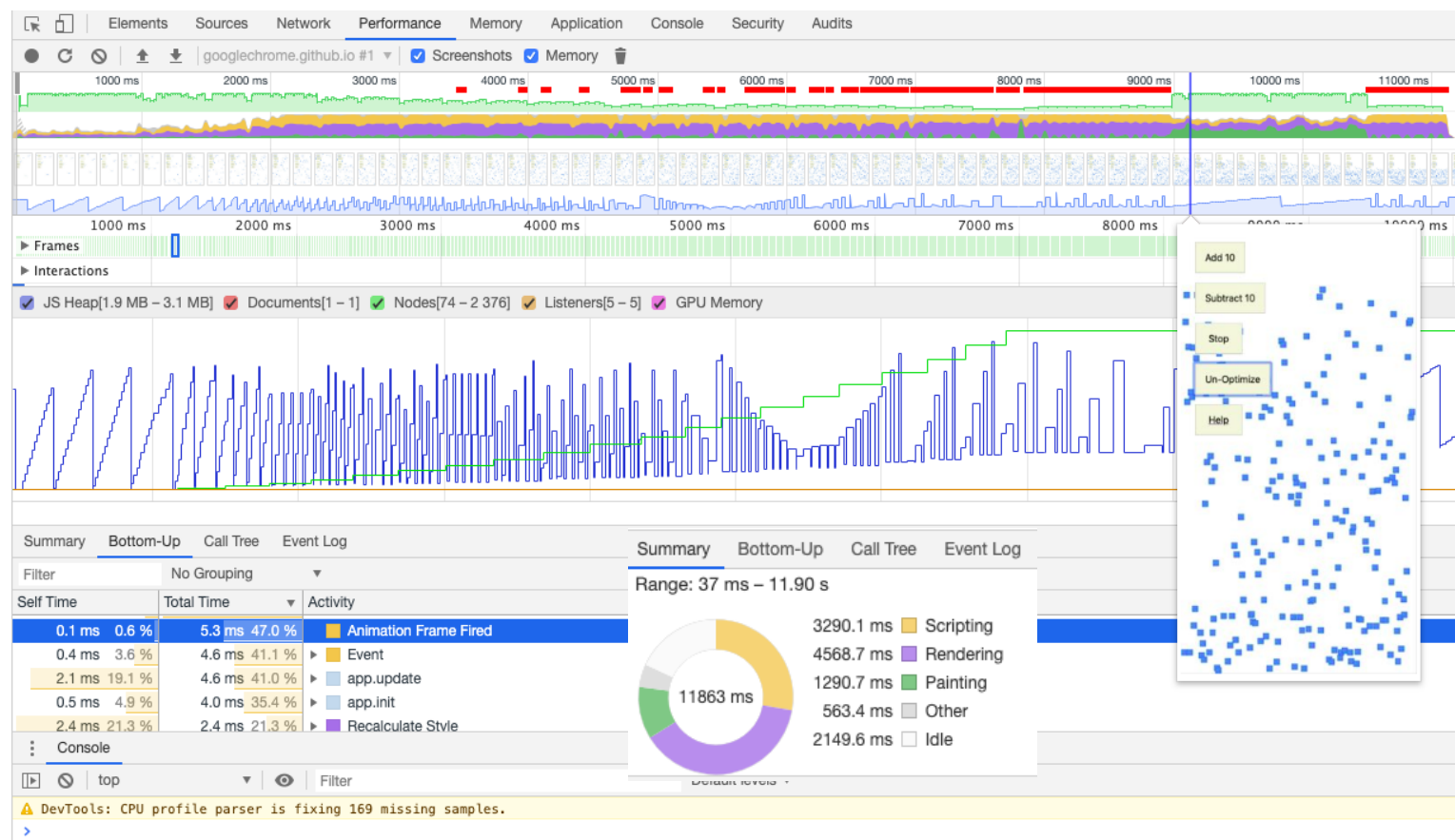
Performance > CPU > 2 x Slowdown



Example with Google Chrome

Profile result:

Frames per Second →
CPU →



Bottleneck →

Other tools for browser

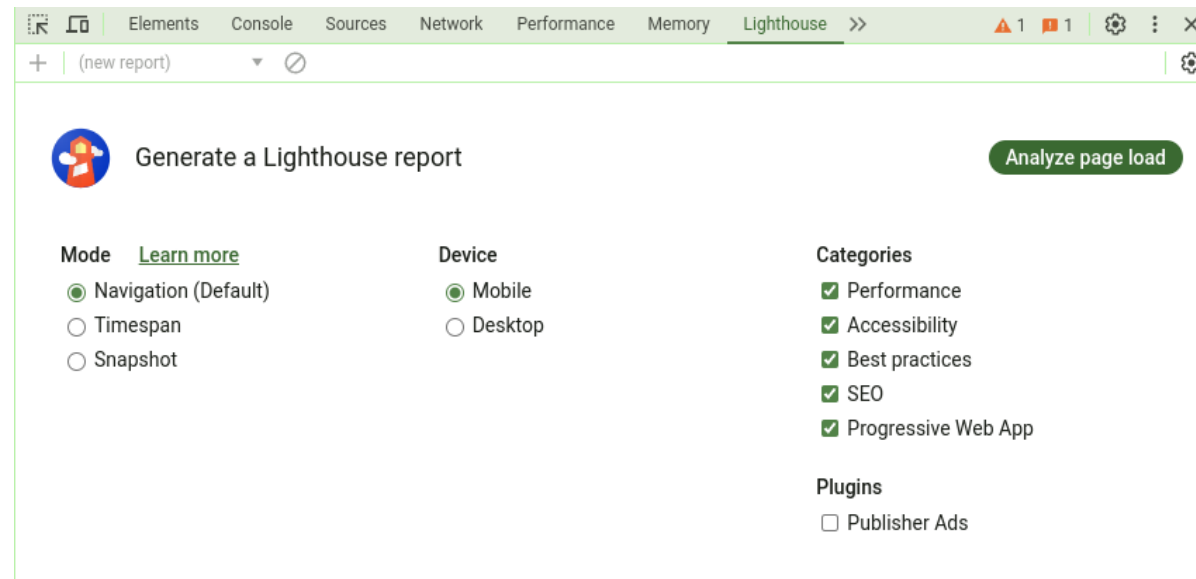
RAIL model:

Response, Animation, Idle, Load

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

<https://webpagetest.org/easy>

Lighthouse (with Chrome)





Server side monitoring

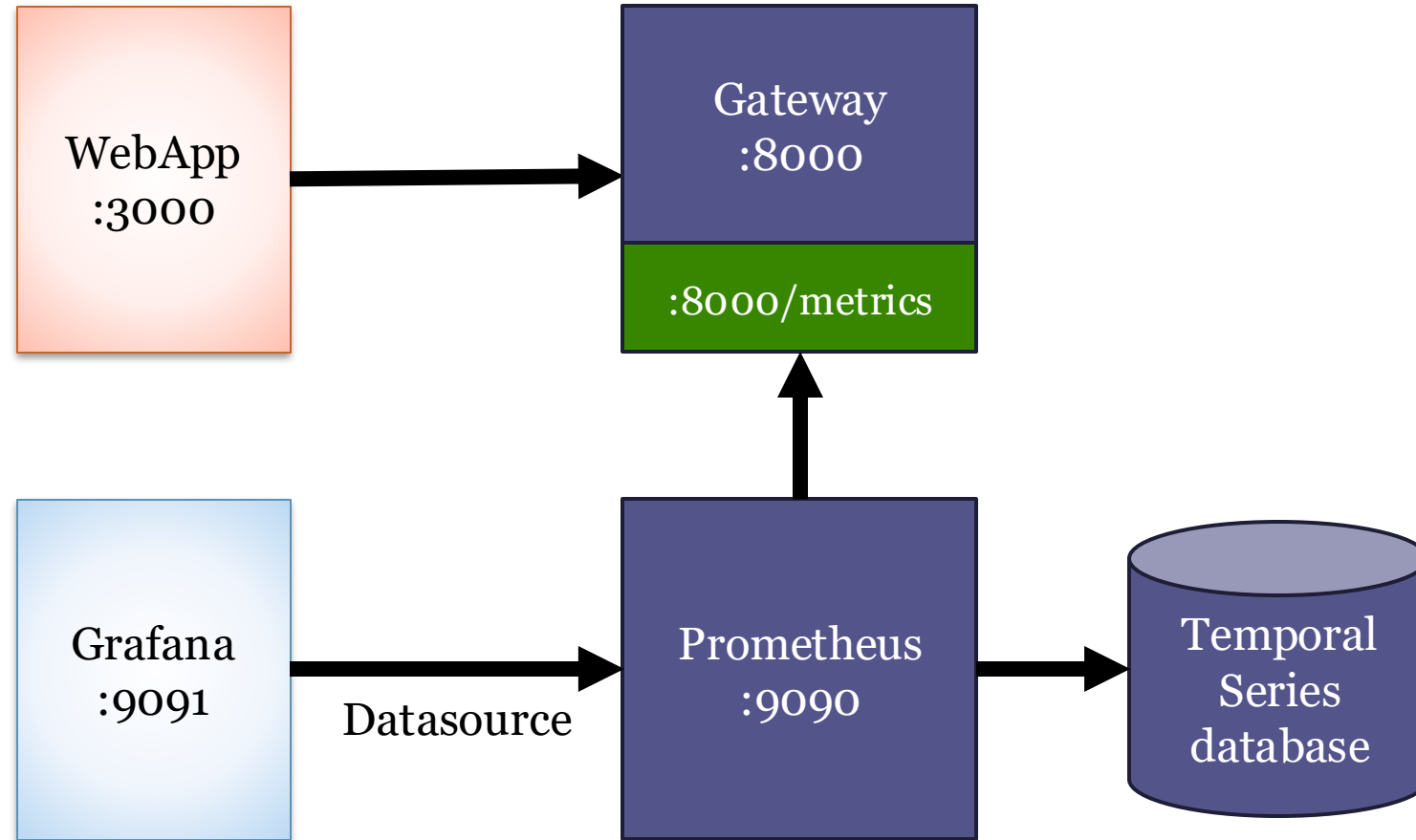
Cloud platforms like Azure provide monitoring solutions

Also available in Google Cloud, Amazon AWS, Alibaba Cloud...

In the case of Azure: [Azure Monitor](#)

We can also set up our own monitoring solution

Typical software: *Prometheus* and *Grafana*



Server side monitoring

We use a library to extract metrics from gateway service

npm install prom-client express-prom-bundle

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

If we launch the userservice, in */metrics* we can see raw data can be used by Prometheus to store it and by Grafana to plot nice charts

We can choose which metrics to measure [\[doc\]](#)



Server side monitoring

- Grafana cannot use this data directly, we need Prometheus

- Prometheus retrieves data exposed by a service (e.g. userservice) and stores it in a time series database so it can be consumed by Grafana
- We configured a docker image [prom/prometheus] with a single file

```
global:
  scrape_interval: 5s

scrape_configs:
  - job_name: 'users-service'
    static_configs:
      - targets: ['users:3000']
```

Server side monitoring

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



Example of Real Grafana Dashboards

<https://grafana.wikimedia.org/>

Links

Monitoring & 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](https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/timeline-tool#profile-js)