Universidad de Granada

Doble Grado en Ingeniería Informática y Matemáticas

Ingeniería de Servidores (Prácticas)

Monitorización de API Web

Autor:
Jesús Muñoz Velasco

Curso 2024-2025

Índice

1. Enunciado

La aplicación empleada en apartado anterior para la prueba de carga, expone en el path "/metrics" los indicadores de NodeJS para Prometheus. Para más información, el exporter de Prometheus de la API Web se ha generado empleando los componentes estandar: prom-client y express-prom-bundle.

Cree un nuevo Dashboard con algunas de las métricas expuestas. Para el dashboard emplee como nombre su nombre y apellidos en CamelCase seguido del sufijo API. Por ejemplo, anaTorrentRamonetAPI. Todos los paneles creados se presentarán con un título que contenga las iniciales del alumno/a. Siguiendo con el ejemplo anterior: %Memoria (ATR).

Cree monitores para las siguientes métricas:

- **)** Tiempos de respuesta de los endpoints de la API (`http_request_duration_seconds_bucket`)
- **)** Memoria disponible (`nodejs_heap_size_total_bytes`) vs la usada actualmente (`nodejs_heap_size_used_bytes`)
- **)** Uso de CPU (`process_cpu_seconds_total`)

Realice una memoria de prácticas en la que se ponga de manifiesto la ejecución de la prueba de carga diseñada para Jmeter y se aprecie el efecto de la misma en los monitores anteriormente descritos.

2. Resolución

2.1. Configuración Inicial

Para comenzar el ejercicio se ha copiado el directorio del ejercicio anterior y se ha metido en una carpeta llamada `Grafana_Prometheus_docker` en el cual se han cambiado algunos archivos de configuración para que prometheus pueda monitorizar la API Web. La red de directorios resultante ha sido la siguiente

```
Grafana_Prometheus_docker
  |-grafana_data
  |-prometheus_data
  |-docker-compose.yml
  |-prometheus.yml
```

Donde el archivo `docker_compose.yml` contiene la siguiente configuración:

```
---
version: "3"
services:
  prometheus:
    image: prom/prometheus:v2.50.0
    extra_hosts:
      - "host.docker.internal:host-gateway"
    ports:
      - 9090:9090
    volumes:
      - ./prometheus_data:/prometheus
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    command:
    - "--config.file=/etc/prometheus/prometheus.yml"
  grafana:
    image: grafana/grafana:9.1.0
    ports:
    - 4000:3000
    volumes:
    - ./grafana_data:/var/lib/grafana
    depends_on:
    - prometheus
```

Y al archivo `prometheus.yml` se le ha añadido un job para poder monitorizar la API:

```
---
global:
  scrape_interval: 5s

scrape_configs:
  - job_name: "prometheus_service"
    static_configs:
      - targets: ["prometheus:9090"]
```

```
- job_name: "API_Web"
  static_configs:
    - targets: ['host.docker.internal:3000'] # Acceso a API
       Web
```

De esta forma podemos conectar los componentes necesarios para la realización del ejercicio. Si volvemos al directorio de jMeter y lanzamos el docker con la API:

```
sudo docker compose up
```

Y hacemos lo mismo en el directorio `Grafana_Prometheus_docker` para activar grafana y prometheus tenemos ya la configuración terminada. Se habilitan los siguientes puertos:

- **)** `localhost:3000`: API de la Web de la ETSIIT
- **)** `localhost:9090`: Prometheus monitorizando la API de la Web de la ETSIIT.
- **)** `localhost:4000`: Grafana

Y ya podremos comenzar con el ejercicio

2.2. Monitorización

En primer lugar crearemos los paneles que se piden en Grafana. Para ello abrimos Grafana y le especificamos en la configuración que coja datos de Prometheus (especificando el `Data Source`).

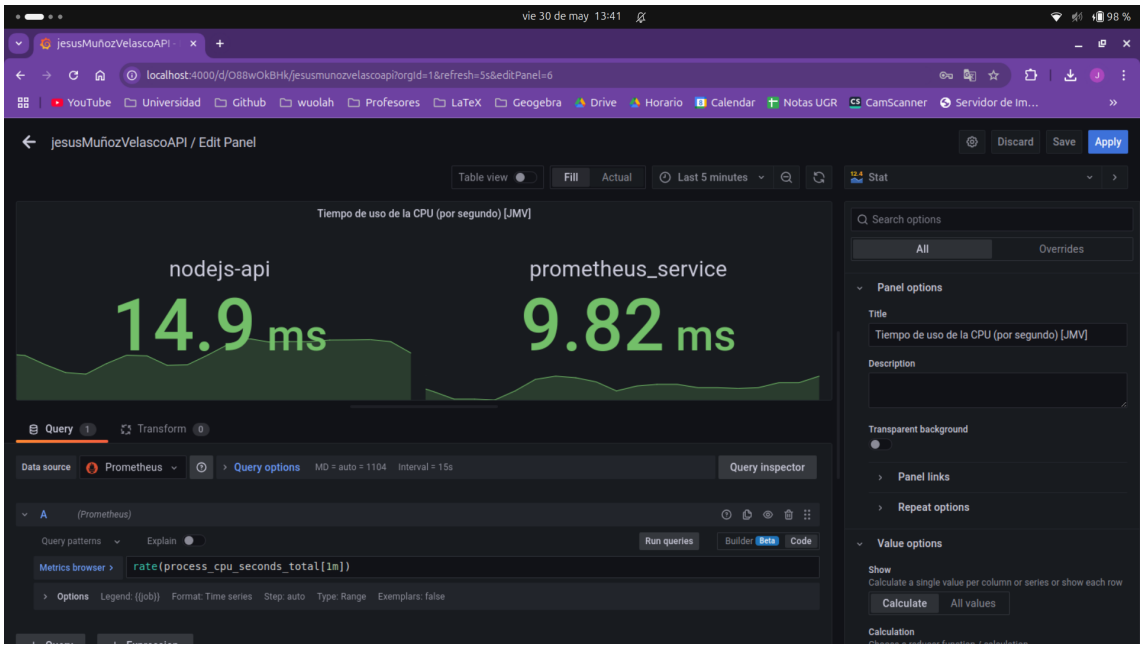
Creamos ahora un nuevo Dashboard (en mi caso lo cree vacío solo para añadir los paneles requeridos). Dentro del Dashboard pasamos a crear los siguientes paneles:

2.2.1. Tiempo de Uso de la CPU

En el código del Query añadimos:

```
rate(process_cpu_seconds_total[1m])
```

Y conseguimos el tiempo de uso de la CPU por segundo. Modificando un poco las leyendas y los estilos, poniéndolo de tipo `Stat` obtenemos el siguiente resultado (img37):

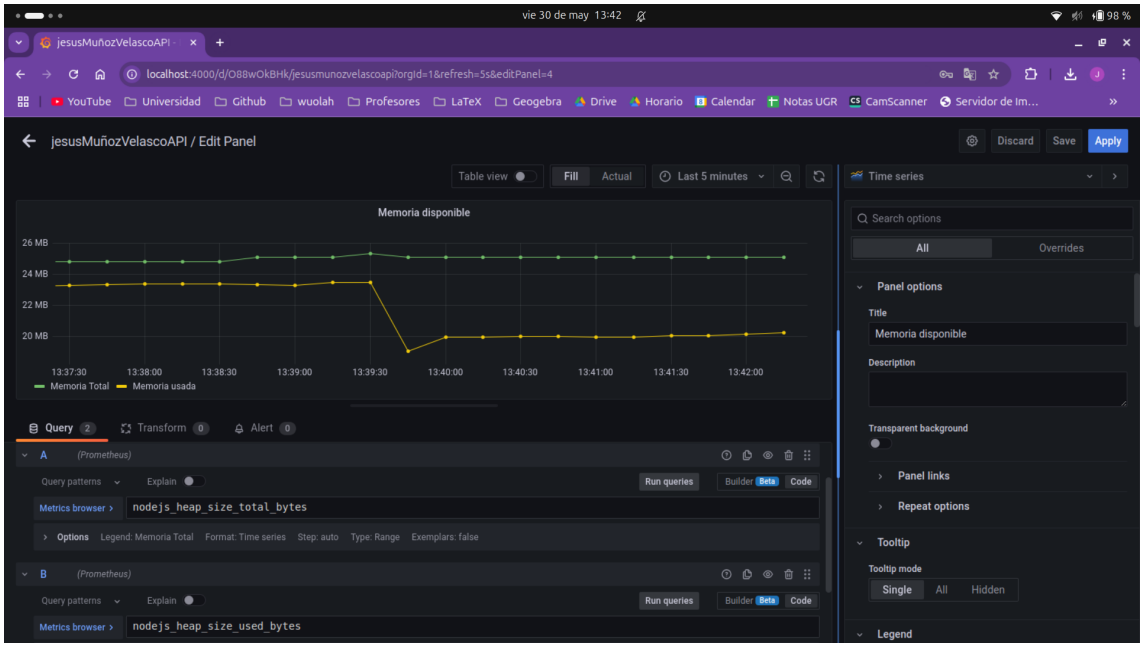


2.2.2. Memoria disponible

Añadimos ahora dos Querys y ponemos:

```
nodejs_heap_size_total_bytes # En la primera
nodejs_heap_size_used_bytes  # En la segunda
```

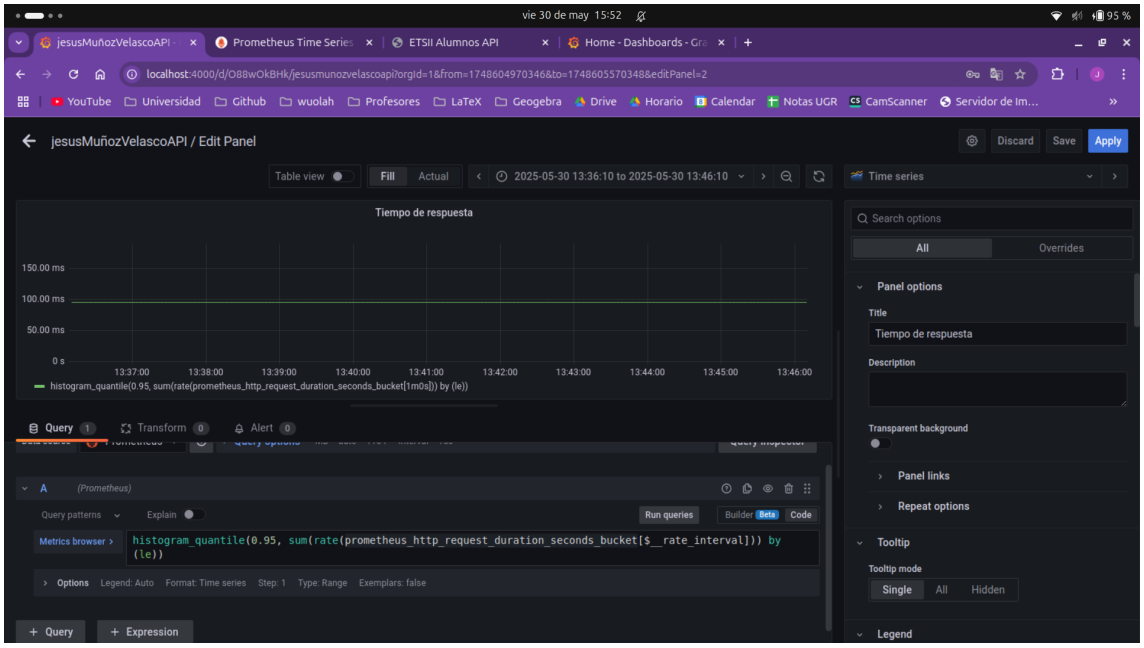
Y conseguimos dos líneas temporales que indican la memoria usada y la memoria total en cada caso (img38).



2.2.3. Tiempos de respuesta de los endpoints de la API

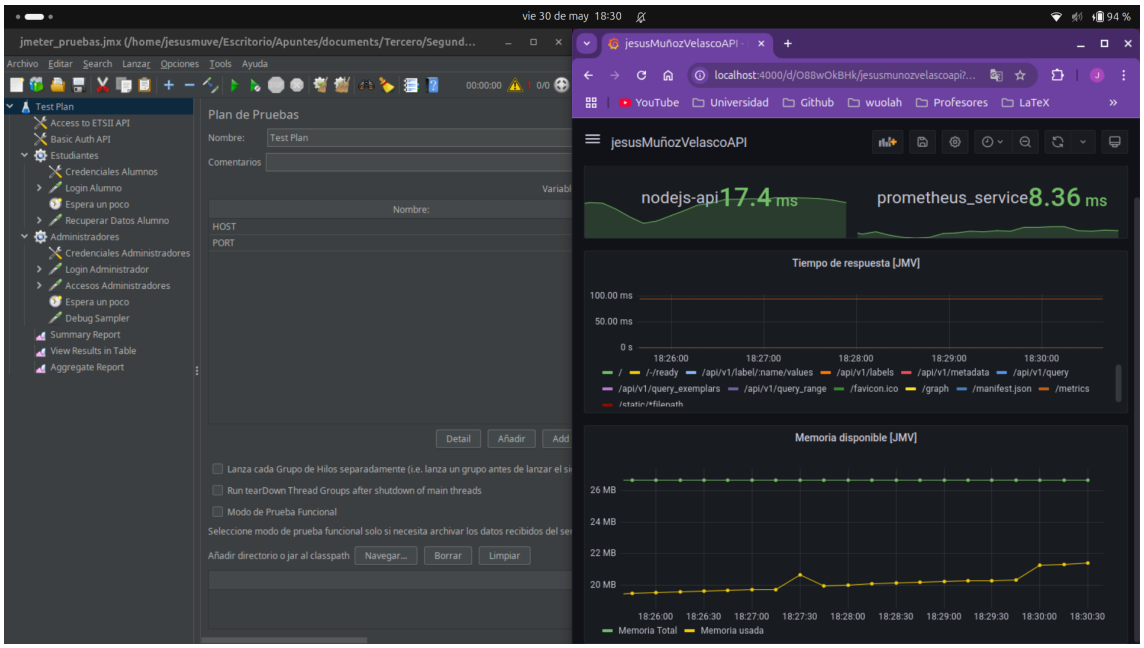
Añadimos ahora (img39):

```
histogram_quantile(0.95, sum(rate(
   prometheus_http_request_duration_seconds_bucket[
   $__rate_interval])) by (le))
```



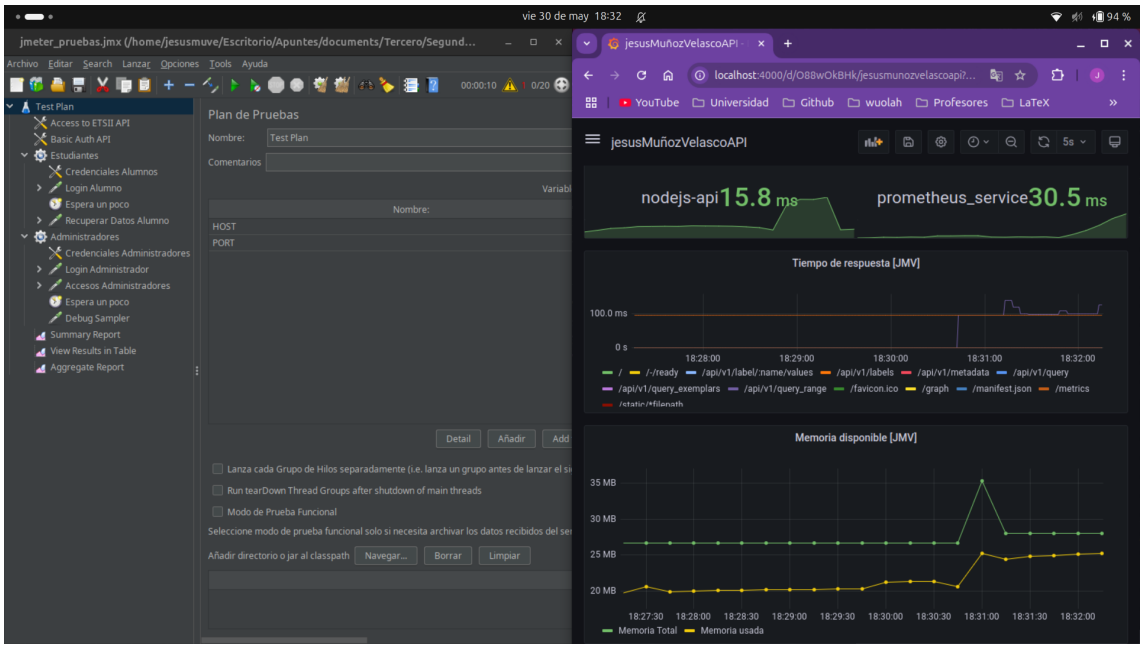
Ya podemos pasar a hacer la prueba de jMeter. Como jMeter estaba configurado para acceder a la API a través de `localhost:3000` no tendremos que modificar nada en su configuración, simplemente ejecutar la prueba de carga.

Al ejecutarlo tenemos inicialmente una situación como la siguiente (img40):



donde todo el sistema monitorizado parece en un estado de uso "bajo".

Al ejecutar la prueba tenemos el siguiente resultado (img41):



Donde se aprecia perfectamente un pico en las medidas tanto de tiempo de uso de CPU como de memoria usada y de tiempo de acceso.