**Curso:** PROGRMACION BACKEND

**Comision:** 30975

**Alumno:** Matias Herreros

**Entregable #:** 15 - logs, profiling & debug

**Repositorio:** <https://github.com/MHerreros/BackendCoderhouse/tree/entregable15>

Contenido

[PROFILING DE SERVIDOR: 2](#_Toc111904945)

[Profiling con **‘—prof’ de node.js** 3](#_Toc111904946)

[Profiling con **‘—prof’ de node.js** utilizando **Artillery** 4](#_Toc111904947)

[Profiling con **‘—prof’ de node.js** utilizando **Autocannon** 5](#_Toc111904948)

[Perfilamiento del servidor con el **modo inspector de node.js**—inspect. 6](#_Toc111904949)

[Diagrama de flama con 0x 7](#_Toc111904950)

[Compresión con GZip 8](#_Toc111904951)

[Anexo: 9](#_Toc111904952)

# PROFILING DE SERVIDOR:

Según lo solicitado en la consigna del entregable, se realizaron las siguientes 3 pruebas sobre la ruta “/info” del archivo “server.js” que se encuentra en el repositorio del entregable ([link to repo](https://github.com/MHerreros/BackendCoderhouse/tree/entregable15)). Sobre dicha ruta se incorporó/elimino un “console log” para ver la diferencia en el análisis.

1. Perfilamiento del servidor con—**prof de node.js**
   1. Utilizar como test de carga Artillery en línea de comandos, emulando 50 conexiones concurrentes con 20 request por cada una.
   2. Luego utilizar Autocannon en línea de comandos, emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos.
2. Perfilamiento del servidor con el **modo inspector de node.js**—inspect.
3. Diagrama de flama con 0x, emulando la carga con Autocannon con los mismos parámetros anteriores.
4. Compresión con GZIP.

## Profiling con **‘—prof’ de node.js**

* Con Console Log:

CON console log


* Sin Console Log:

SIN Console Log


Sin el ‘Console Log’ en la ruta ‘/info’ se tiene aproximadamente un ahorro del 50% de ticks en eventos relacionados a Java Script.

## Profiling con **‘—prof’ de node.js** utilizando **Artillery**

* Con Console Log:

Patrón de fondo

Descripción generada automáticamente

* Sin Console Log:

Patrón de fondo

Descripción generada automáticamente

Según los resultados de Artillery, el tiempo medio de procesamiento de solicitudes a ‘/info’ fue levemente menor cuando no se utilizó el ‘console log’.

## Profiling con **‘—prof’ de node.js** utilizando **Autocannon**

* Con Console Log:

Interfaz de usuario gráfica

Descripción generada automáticamente

* Sin Console Log:

Interfaz de usuario gráfica

Descripción generada automáticamente

Se observa que sin Console Log se procesan levemente mas requests por segundo (97.85 vs 93.65 promedio). La latencia también es algo menor (983.9 vs 1040.2 promedio).

## Perfilamiento del servidor con el **modo inspector de node.js**—inspect.

La siguiente imagen corresponde al perfilamiento del servidor (con console log en la ruta /info) utilizando el modo inspect de Chrome:

Texto

Descripción generada automáticamente

Se puede observar que el console log aporta una perdida de tiempo en el procesamiento de los requests de 13.1 ms. Esta perdida se da ante una prueba hecha con autocannon con una concurrencia de 100 durante 20 seg.

* Sin console log:

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

Descripción generada automáticamente

* Con console log:

Tabla, Excel

Descripción generada automáticamente

Puede observarse que, a nivel general, el proceso que mas tiempo toma es el de “consoleCall”. Revisando mas a detalle este proceso se observa que aquí están incluidos los console logs realizados por el servidor (los realizados por log4js y los console log nativos propios del script). De la comparación de las tablas de arriba, puede notarse que el tiempo de procesamiento asociado a los console logs casi se duplica al añadir el console log en la ruta /info.

## Diagrama de flama con 0x

Aplicación

Descripción generada automáticamente con confianza baja

En la imagen de arriba pueden observarse 2 picos de actividad. El primero se corresponde a una prueba con ‘autocannon’ hacia la ruta ‘/info’ sin ‘console log’ y el segundo corresponde a la misma prueba, pero hacia la ruta ‘/info’ con ‘console log’. Si bien es difícil de apreciar en esta captura, la altura de la línea *con ‘console log’* es mayor que la prueba donde no se tiene dicho ‘console’. En ambos casos la consulta a ‘/info’ y el ‘console log’ (dependiendo del caso) se encuentran bien arriba en el grafico de flama, es decir que bloquean momentáneamente los procesos de abajo. Sin embargo, el bloqueo es por muy poco tiempo por lo que no se ve un gran impacto en el grafico (mesetas).

## Compresión con GZip

* Comprimido:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* Sin Comprimir:

Captura de pantalla de computadora

Descripción generada automáticamente

Puede observarse que, al no comprimir el request a la ruta /info se transmiten 8.2 kB de información mientras que al comprimirse solo se transmiten 2.0 kB de información. Esto es un ahorro de un 75% en el trafico de información. En definitiva, cuando el volumen de información lo amerita, es recomendable el uso de ‘compression’.

## Anexo:

Toda la documentación respaldatoria se encuentra cargada en el repositorio público. El link a dicho repositorio se encuentra en la primera página del informe.