



# ESCALADO EN, MONOLÍTICO

Eddie Girón

Julio Ruíz



# ¿CÓMO ESCALAMOS UN MONOLITO?

Puntos a mejorar para escalar una aplicación monolítica:

- Mejora de capacidad de carga
- Optimización general de código
- Recursos adicionales
- Implementación de técnicas de escalabilidad
- Refactorización
- Migración a otra arquitectura

Escalar una arquitectura monolítica puede ser más complicado y caro que otras arquitecturas.

# ¿PUEDE ESCALAR VERTICALMENTE?

El escalado vertical es una forma directa de manejar un aumento en la carga; sin embargo, tiene límites físicos y financieros. Eventualmente, agregar más recursos se vuelve costoso o técnicamente inviable.

- Aumento de recursos de hardware
- Utilización de Caché

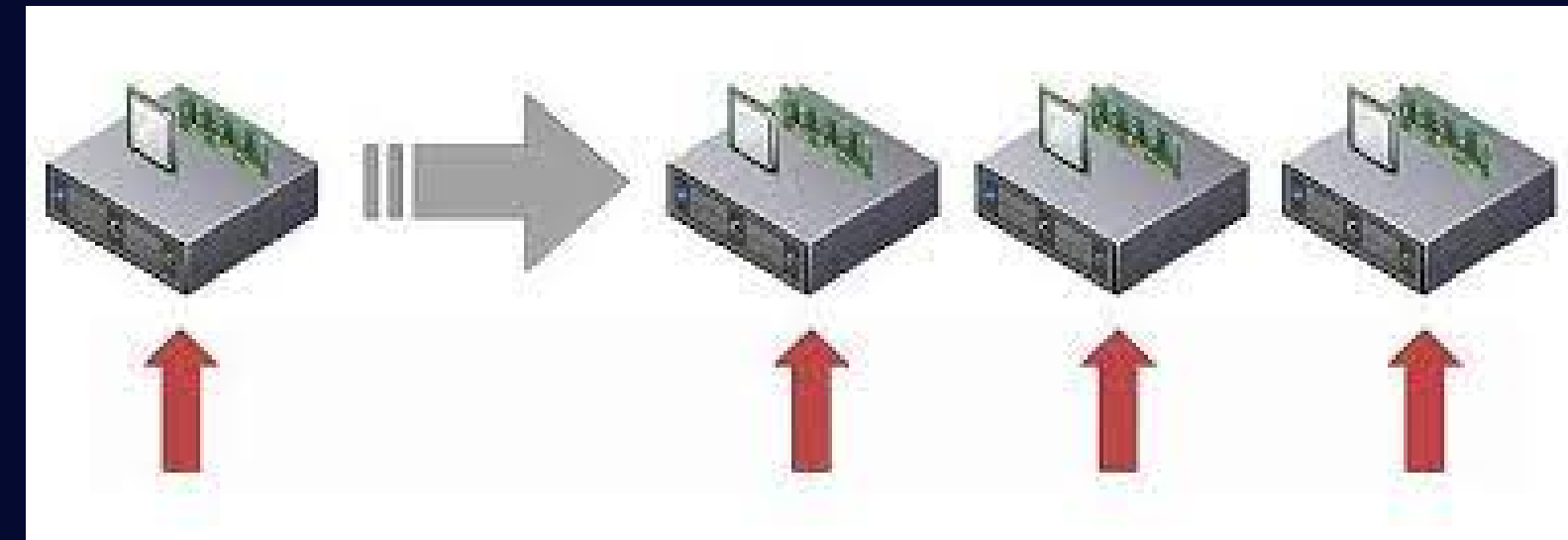


# ¿PUEDE ESCALAR HORIZONTALMENTE?

Aumento en la capacidad de una aplicación distribuyendo la carga de trabajo entre múltiples instancias de la misma.

Escalamiento en aplicación de log-in:

- Persistencia de datos
- Automatización de despliegues



# ¿ARQUITECTURA MONOLÍTICA CON ORLEANS?

Puede hacerse, sin embargo no se aprovecharía el potencial completo de Orleans, aunque si se hace con fines de escalabilidad futura puede ser conveniente.

Git: [https://github.com/Edd1enator/SimpleLog\\_In.git](https://github.com/Edd1enator/SimpleLog_In.git)

# HIPOTESIS

Hipótesis de Rendimiento: La aplicación maneja un número específico de usuarios concurrentes (por ejemplo, 1400 usuarios simultáneos) sin degradar significativamente el rendimiento (por ejemplo, el tiempo de respuesta promedio permanecerá por debajo de 2 segundos).

# RECURSOS

- 2 CPU
- 4 RAM

# /register

Thread Group.jmx (C:\Users\JULIORUIZ\Desktop\Thread Group.jmx) - Apache JMeter (5.6.3)

File Edit Search Run Options Tools Help

00:00:10 0 0/700

Pruebas Rendimiento

- Thread Group
  - HTTP Request
    - View Results Tree
    - Summary Report**
    - Response Time Graph
  - Thread Group 2
    - HTTP Request
      - View Results Tree
      - Summary Report
      - Response Time Graph

### Summary Report

Name: Summary Report

Comments:

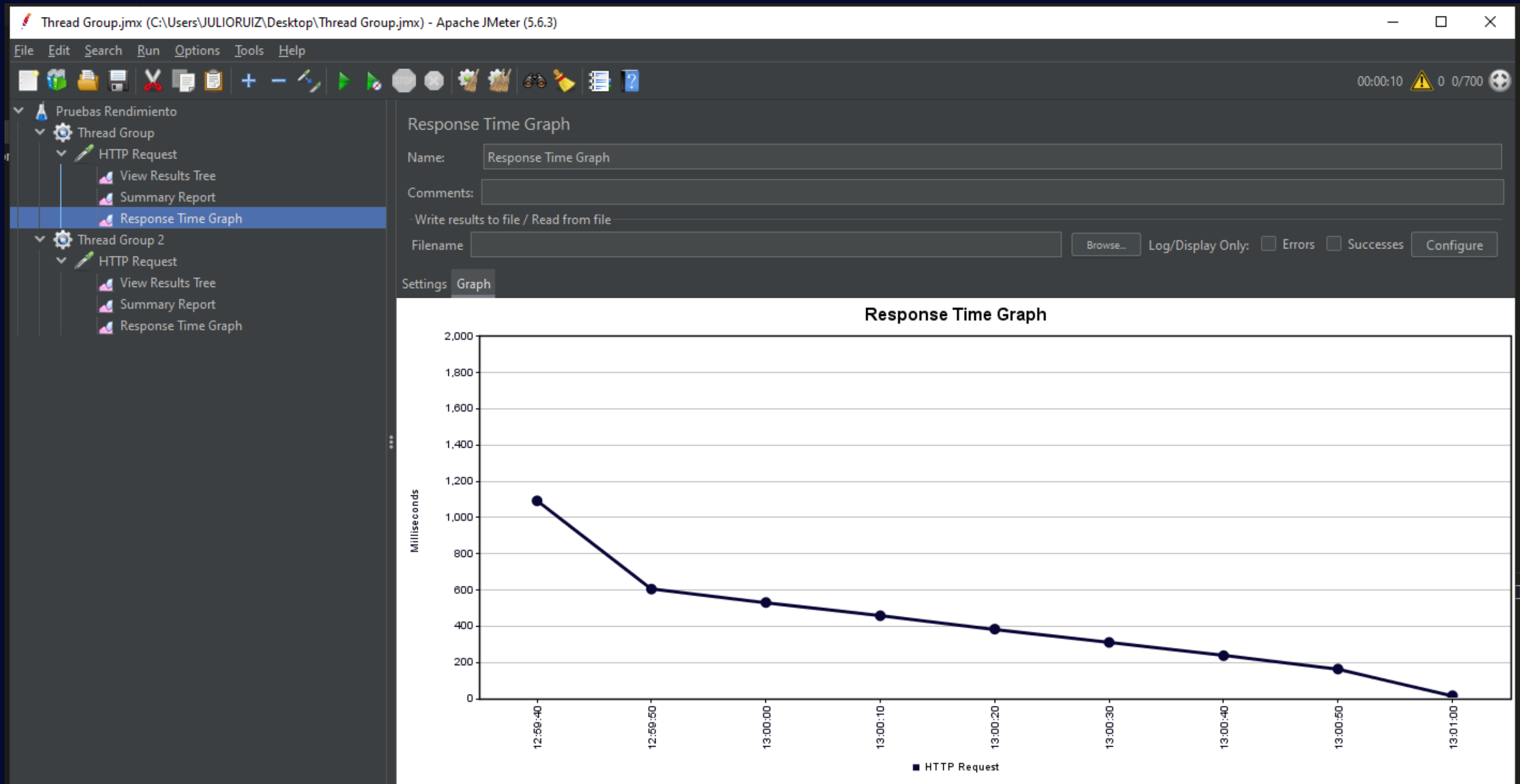
Write results to file / Read from file

Filename:   Log/Display Only: ☐ Errors ☐ Successes

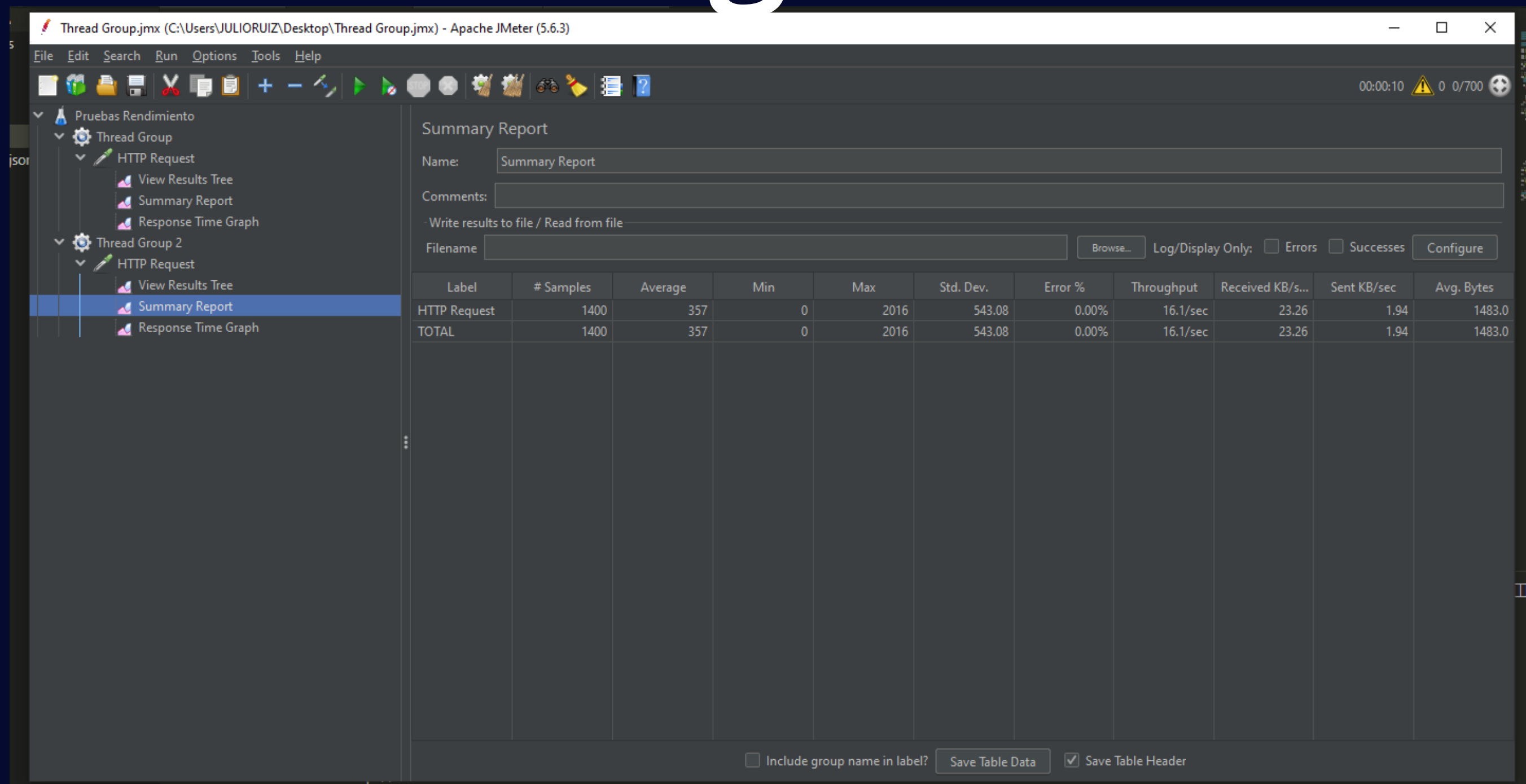
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
HTTP Request	1400	347	1	2059	539.11	0.00%	16.1/sec	23.30	1.99	1486.0
TOTAL	1400	347	1	2059	539.11	0.00%	16.1/sec	23.30	1.99	1486.0

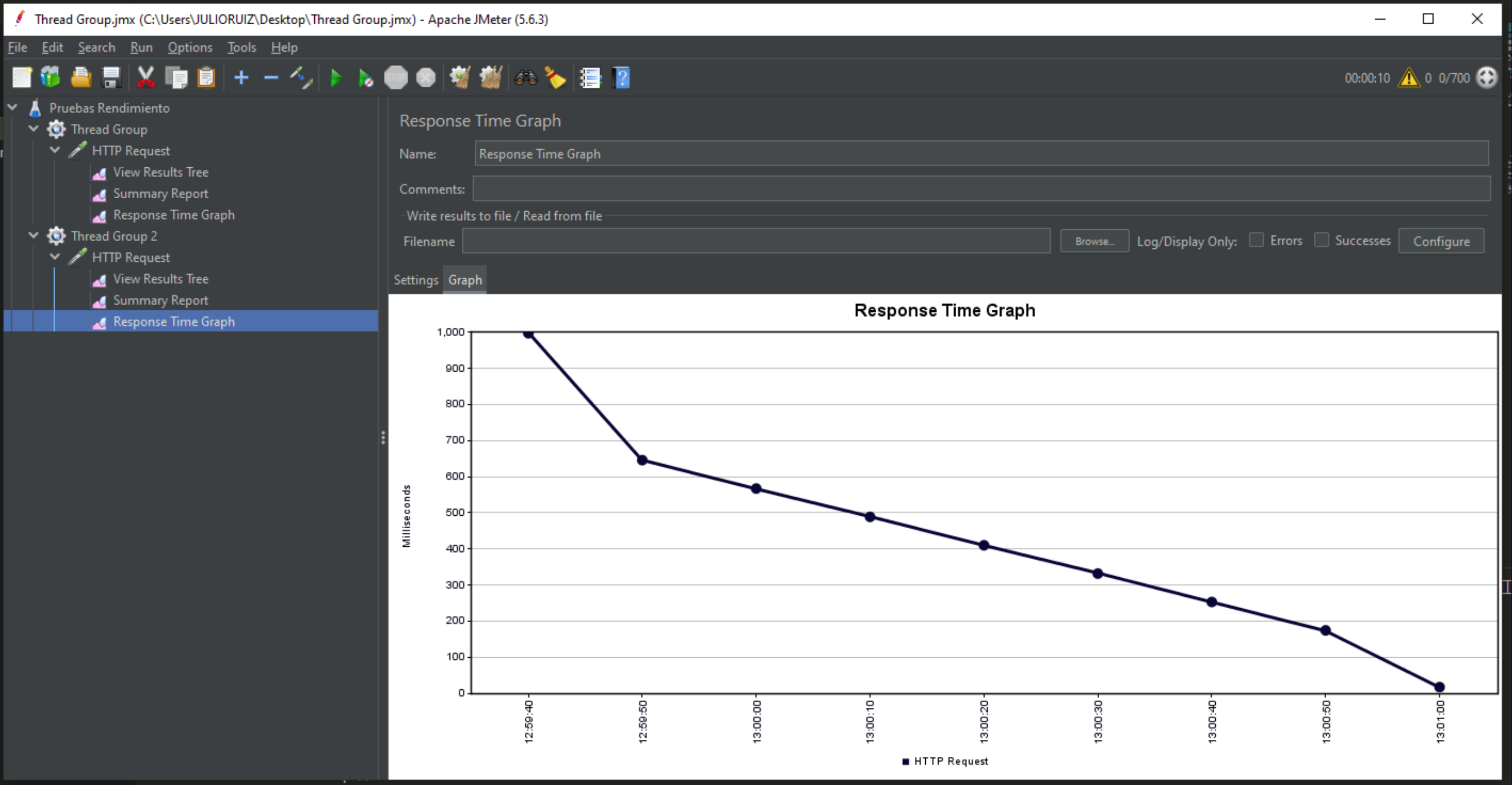
☐ Include group name in label?  ☒ Save Table Header





# /login














# RECURSOS

- 1 CPU
- 2 RAM

## Virtual Machine Settings

Hardware Options

Device	Summary
 Memory	4 GB
 Processors	1
 Hard Disk (NVMe)	60 GB
 CD/DVD (SATA)	Auto detect
 Floppy	Auto detect
 Network Adapter	NAT
 USB Controller	Present
 Sound Card	Auto detect
 Display	Auto detect

Add...

Remove

## Memory

Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

Memory for this virtual machine: 2048 MB

128 GB -  
64 GB -  
32 GB -  
16 GB -  
8 GB -  
4 GB -  
2 GB -  
1 GB -  
512 MB -  
256 MB -  
128 MB -  
64 MB -  
32 MB -  
16 MB -  
8 MB -  
4 MB -

Maximum recommended memory  
(Memory swapping may  
occur beyond this size.)  
27.3 GB

Recommended memory  
2 GB

Guest OS recommended minimum  
2 GB

OK

Cancel

Help

# /register primera prueba

Thread Group.jmx (C:\Users\JULIORUIZ\Desktop\Thread Group.jmx) - Apache JMeter (5.6.3)

File Edit Search Run Options Tools Help

00:00:26 0 0/700

Pruebas Rendimiento

- Thread Group
  - HTTP Request
    - View Results Tree
    - Summary Report
    - Response Time Graph
  - Thread Group 2
    - HTTP Request
      - View Results Tree
      - Summary Report
      - Response Time Graph

Summary Report

Name: Summary Report

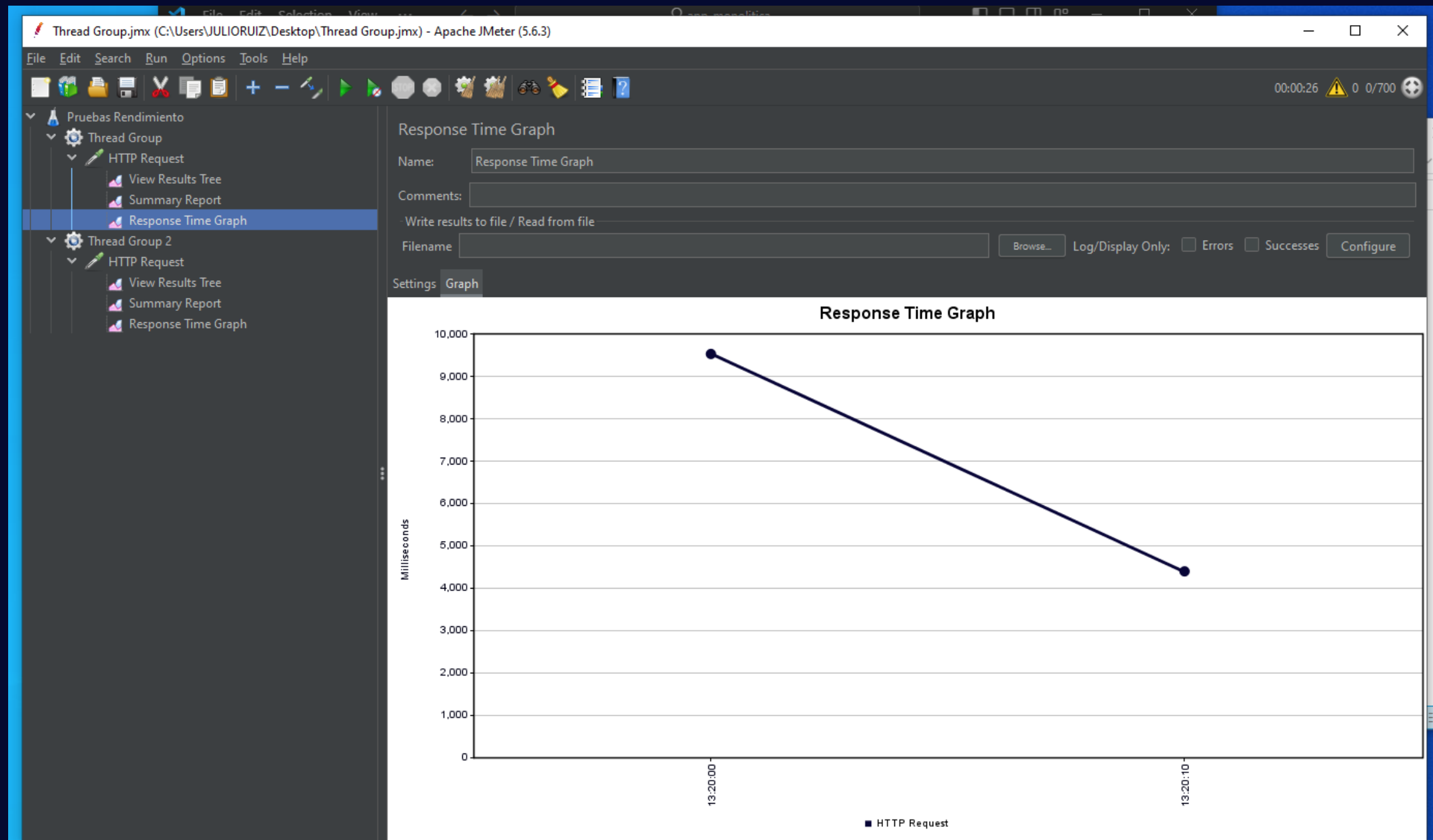
Comments:

Write results to file / Read from file

Filename:   Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	700	6000	3	19194	5559.75	17.14%	27.1/sec	45.01	2.79	1699.9
TOTAL	700	6000	3	19194	5559.75	17.14%	27.1/sec	45.01	2.79	1699.9

☐ Include group name in label?  ☒ Save Table Header



# /login primera prueba

Thread Group.jmx (C:\Users\JULIORUIZ\Desktop\Thread Group.jmx) - Apache JMeter (5.6.3)

File Edit Search Run Options Tools Help

00:00:26 0 0/700

Pruebas Rendimiento

- Thread Group
  - HTTP Request
    - View Results Tree
    - Summary Report
    - Response Time Graph
  - Thread Group 2
    - HTTP Request
      - View Results Tree
      - Summary Report
      - Response Time Graph

Summary Report

Name: Summary Report

Comments:

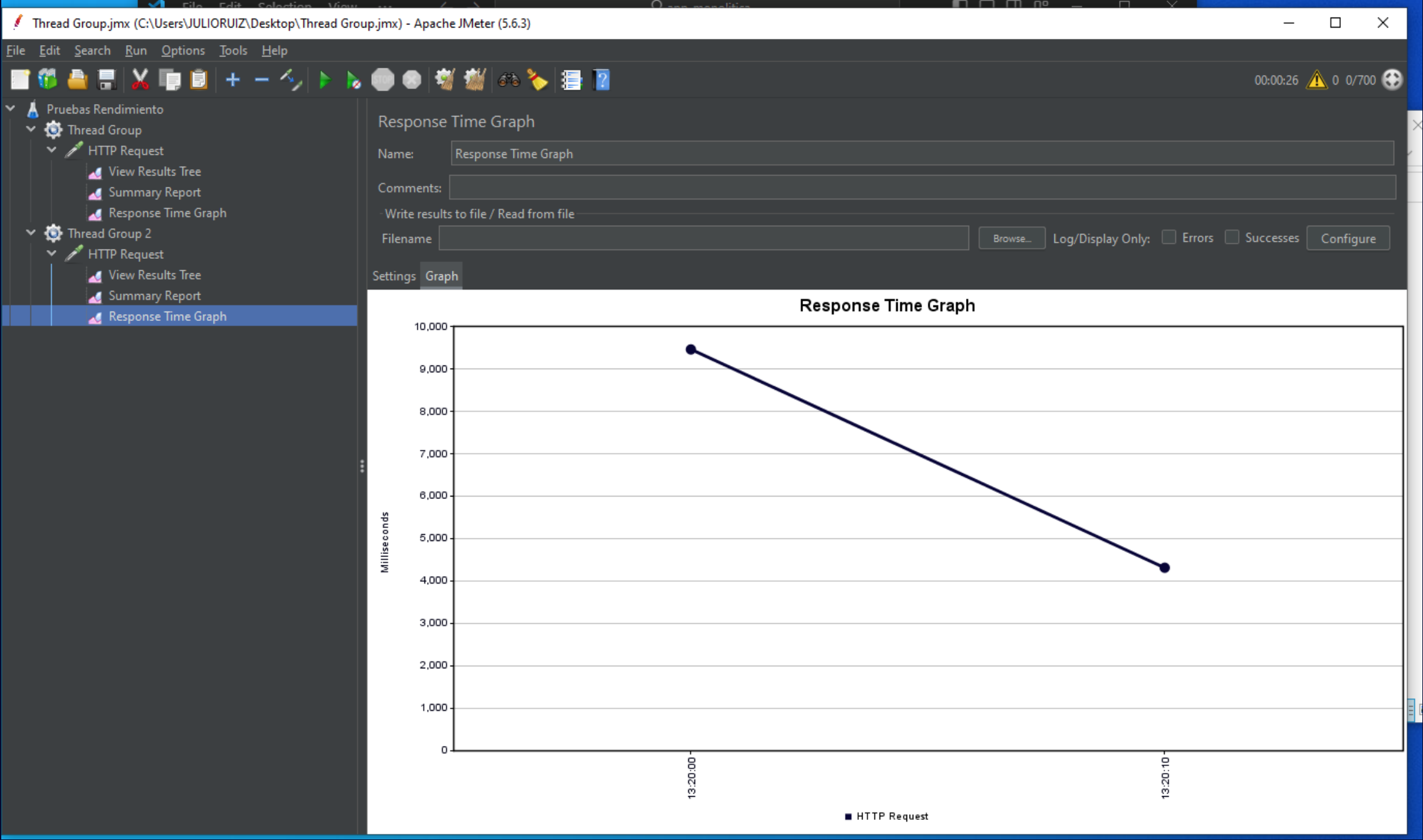
Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	700	5941	6	19190	5569.02	16.43%	27.3/sec	45.09	2.77	1689.7
TOTAL	700	5941	6	19190	5569.02	16.43%	27.3/sec	45.09	2.77	1689.7

☐ Include group name in label? Save Table Data ☒ Save Table Header





# /register segunda prueba

Thread Group.jmx (C:\Users\JULIORUIZ\Desktop\Thread Group.jmx) - Apache JMeter (5.6.3)

File Edit Search Run Options Tools Help

00:00:25 0 0/700

Pruebas Rendimiento

- Thread Group
  - HTTP Request
    - View Results Tree
    - Summary Report
    - Response Time Graph
  - Thread Group 2
    - HTTP Request
      - View Results Tree
      - Summary Report
      - Response Time Graph

### Summary Report

Name: Summary Report

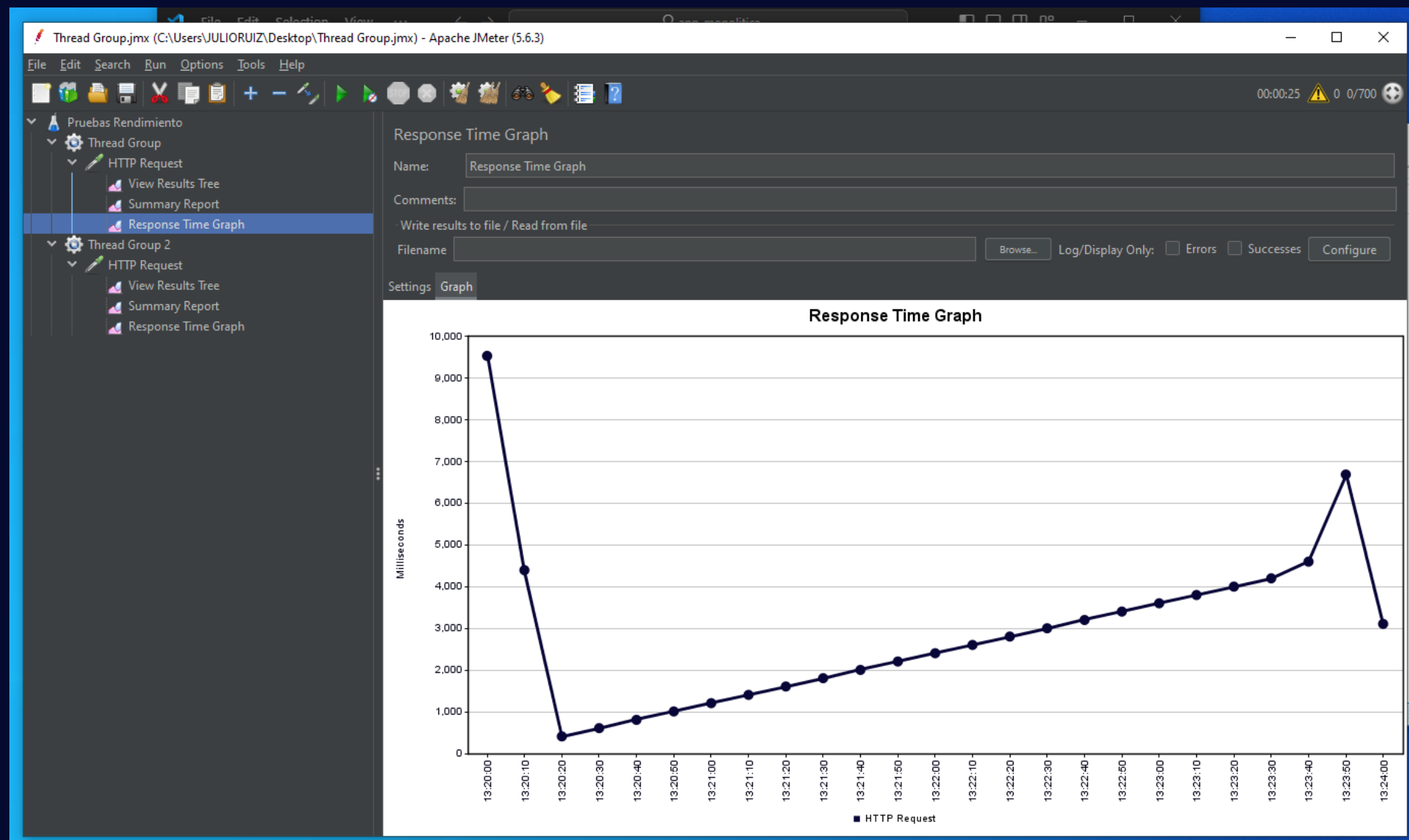
Comments:

Write results to file / Read from file

Filename:   Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	1400	5701	3	19194	5020.71	18.14%	5.6/sec	9.35	0.57	1713.0
TOTAL	1400	5701	3	19194	5020.71	18.14%	5.6/sec	9.35	0.57	1713.0

☐ Include group name in label?  ☒ Save Table Header



# /login segunda prueba

Thread Group.jmx (C:\Users\JULIORUIZ\Desktop\Thread Group.jmx) - Apache JMeter (5.6.3)

File Edit Search Run Options Tools Help

00:00:25 0 0/700

Pruebas Rendimiento

- Thread Group
  - HTTP Request
    - View Results Tree
    - Summary Report
    - Response Time Graph
  - Thread Group 2
    - HTTP Request
      - View Results Tree
      - Summary Report
      - Response Time Graph

Summary Report

Name: Summary Report

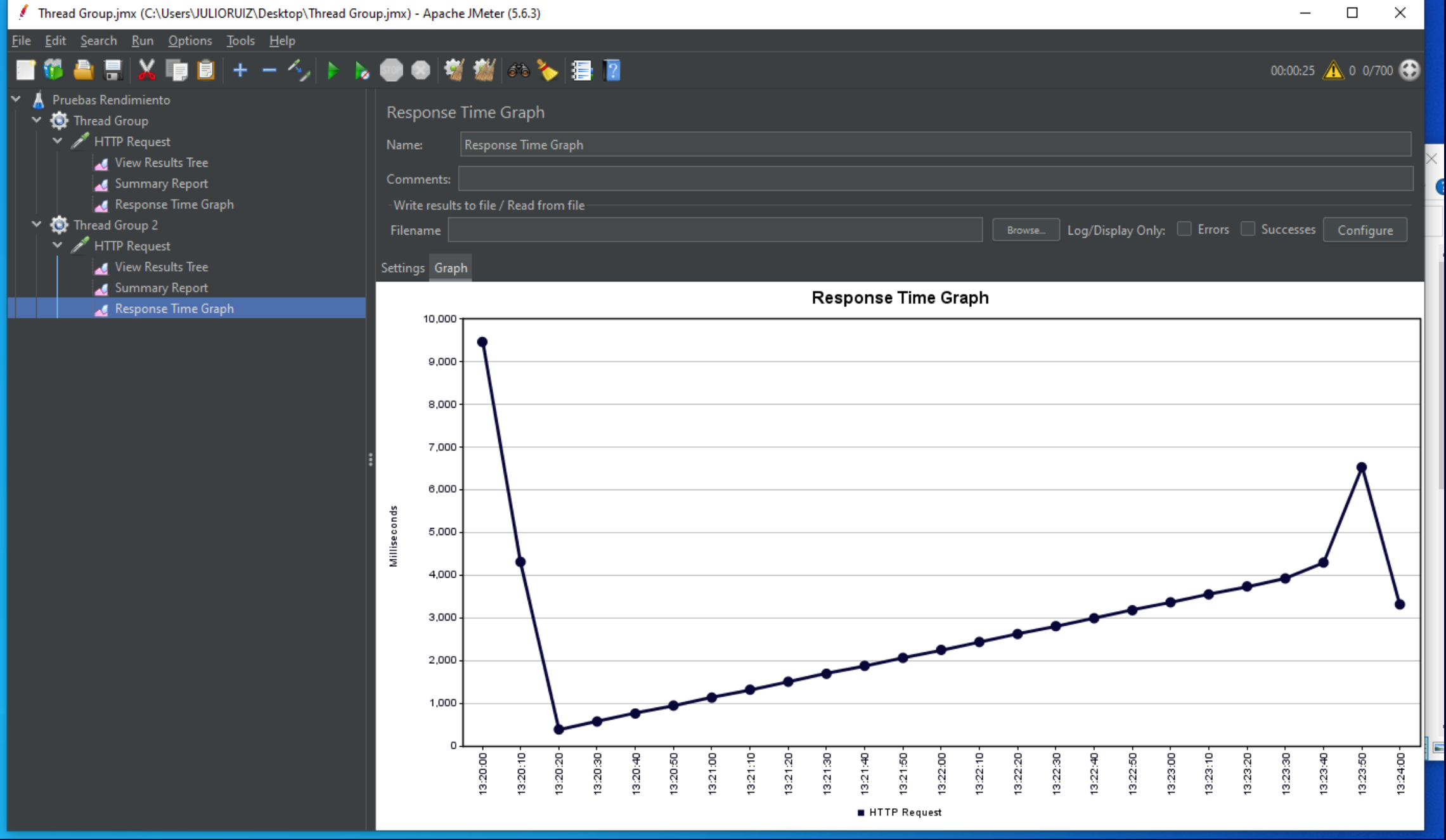
Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
HTTP Request	1400	5636	6	19190	5012.51	17.71%	5.6/sec	9.32	0.56	1705.8
TOTAL	1400	5636	6	19190	5012.51	17.71%	5.6/sec	9.32	0.56	1705.8

☐ Include group name in label? Save Table Data ☒ Save Table Header



# CONCLUSIONES

Con un servidor más robusto (2 CPUs, 4 GB de RAM), la aplicación tomó un promedio de 16.1 segundos para completar operaciones tanto de registro como de login con 1400 usuarios concurrentes. Este tiempo de respuesta es bastante alto, lo que sugiere que la experiencia del usuario podría verse afectada negativamente bajo esta carga. En un entorno de producción, querrías apuntar a tiempos de respuesta inferiores a varios segundos.

Al reducir a la mitad los recursos del servidor (1 CPU, 2 GB de RAM), y también a la mitad la carga de usuarios (700 usuarios), el tiempo de respuesta aumentó significativamente a más de 27 segundos para ambas operaciones. Esto es un deterioro sustancial y señala que la aplicación es posiblemente más sensible a la disponibilidad de CPU que a la memoria, aunque ambos son factores críticos.

- La mejora en el tiempo de respuesta durante la "segunda vuelta" podría sugerir que hay un efecto de calentamiento en juego. Este efecto puede deberse a la caché de la base de datos que se llena, la JIT (Just-In-Time) compilation en el servidor de aplicaciones, o simplemente que ciertas operaciones iniciales pesadas ya se han realizado y no necesitan repetirse.
- Es interesante que, incluso con recursos reducidos, la segunda vuelta tuvo tiempos de respuesta de 5.6 segundos, lo cual es mejor que la primera vuelta con más recursos. Esto podría indicar que la aplicación tiene una etapa inicial pesada o que se beneficia significativamente de la caché o también debido al menor rendimiento del sistema se puede apreciar que existe una mayor cantidad de operaciones fallidas, las cuales provocarían un menor tiempo de ejecución.