Proyecto Final: Análisis de Desempeño con JMeter

Curso: Principio Principios & Prácticas De Desarrollo De Software Orientado A Objetos – C1
Instructor: Gabriel Rodrigo Pedraza Ferreira
Universidad Industrial de Santander

Integrantes

- Sergio Nelson Alberto Gómez Gil - 2214106
- Juan Pablo Ávila Quitián - 2214107
- Juan David Saavedra González – 2214111

Repositorio en GitHub

https://github.com/JuanDavidSaavedra/Proyecto_Software_3.git

Introducción

Este proyecto final se centra en el análisis empírico del comportamiento de una aplicación bajo distintos escenarios de despliegue, utilizando **Docker Compose** y **Kubernetes**.

Nosotros como estudiantes desplegaremos una aplicación en configuraciones que varían desde un único contenedor hasta clústeres de Kubernetes con diferentes niveles de escalabilidad (número de réplicas y nodos).

El objetivo principal es observar y cuantificar métricas de rendimiento clave, como el **tiempo medio de respuesta** y el **throughput**, bajo diferentes cargas, para luego analizar cómo el entorno de despliegue impacta en la aplicación y extraer conclusiones sobre su **escalabilidad** y **eficiencia**.

Objetivo General

Analizar el comportamiento de una aplicación bajo diferentes escenarios de despliegue utilizando herramientas de contenedores como **Docker Compose** y de orquestación como **Kubernetes**, observando métricas de rendimiento y escalabilidad para comprender las implicaciones de cada configuración, especialmente en lo referente al **escalado horizontal**.

Objetivos Específicos

- Desplegar una aplicación utilizando **Docker Compose** en un entorno de una única máquina.
- Desplegar la misma aplicación en un clúster de **Kubernetes** usando uno, dos y tres nodos, variando el número de réplicas de los servicios de la aplicación.
- Utilizar la herramienta **JMeter** para generar carga en la aplicación en cada escenario de despliegue y observar métricas clave como **tiempo medio de respuesta** y **throughput** (tasa de peticiones).
- Observar y registrar el comportamiento de las métricas bajo diferentes cargas y número de réplicas en cada escenario.
- Analizar las diferencias en el comportamiento de la aplicación entre los distintos escenarios de despliegue y las variaciones en el número de réplicas.
- Extraer conclusiones basadas en los datos observados sobre la **escalabilidad**, la **eficiencia de recursos** y las posibles **limitaciones** de cada escenario y configuración de réplicas.

Descripción del Proyecto

Este proyecto de ingeniería de software tiene como objetivo desarrollar una plataforma web eficiente y accesible para la compra de boletas, buscando obtener una experiencia de usuario y una gestión de eventos optimizada. La plataforma permitirá a los usuarios buscar eventos, seleccionar localidades, controlar la cantidad de ventas y vista de informes. A nivel técnico, el desarrollo se basará en tecnologías modernas de desarrollo web para la facilidad de su uso e implementación. El sistema está diseñado para ofrecer una experiencia de usuario fluida, cubriendo tanto las necesidades de los administradores como las de los usuarios finales.

Metodología de Pruebas con JMeter

La configuración que se va a usar para poner a prueba el funcionamiento del despliegue serán las siguientes:

- Threads: 2000
- Ramp-up: 60
- Loop Count: Forever
- Duration: 10 minutos
- Timer entre peticiones: 2s (Constant Timer)

Esta configuración de JMeter es ideal para realizar pruebas de carga intensiva o stress test, ya que simula 2000 usuarios concurrentes accediendo al sistema durante 10 minutos. El ramp-up de 60 segundos distribuye progresivamente la creación de los hilos (usuarios), evitando una sobrecarga repentina y permitiendo observar cómo responde el sistema ante un aumento gradual de tráfico. Al usar un loop infinito con duración definida, se garantiza

que todos los hilos generen solicitudes de manera continua durante todo el período de prueba, lo cual es útil para medir estabilidad y rendimiento sostenido.

Además, establecer un Startup Delay de 2 segundos y controlar la vida útil de los hilos ayuda a sincronizar correctamente el inicio del test con herramientas de monitoreo, y asegura que todos los hilos trabajen durante el mismo intervalo. Esta combinación permite obtener métricas clave como tiempo de respuesta, throughput y tasa de errores bajo condiciones cercanas a un escenario de producción o incluso de sobrecarga, revelando cuellos de botella o límites de capacidad en el sistema evaluado.

Organizando los datos obtenidos en JMeter

Se instalan y se importan las librerías necesarias como:

```
!pip install pandas matplotlib seaborn
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Preparación de los datos

```
#Cargar archives fase 1
file_fase1 = "fase1.csv"
df1 = pd.read_csv(file_fase1)

# Agregar una columna para identificar el número de réplicas
df1["replicas"] = 1

# Combinar los datos en un solo DataFrame
df1_all = df1.copy()

# Cargar los archivos fase 2
file2_1 = "fase2_1replica.csv"
file2_2 = "fase2_2replica.csv"
file2_3 = "fase2_3replica.csv"

df2_1 = pd.read_csv(file2_1)
df2_2 = pd.read_csv(file2_2)
df2_3 = pd.read_csv(file2_3)
```

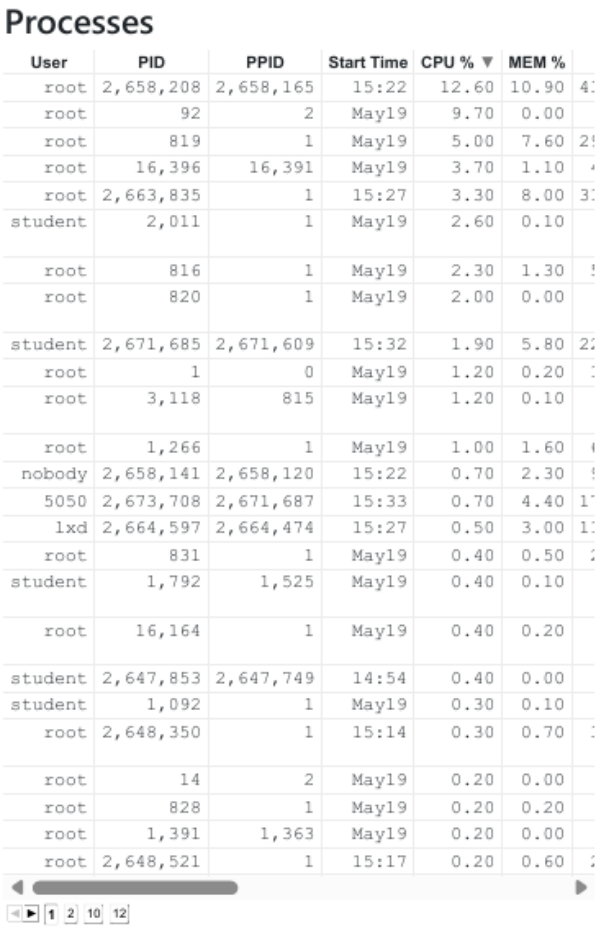
```
# Agregar una columna para identificar el número de réplicas
df2_1["replicas"] = 1
df2_2["replicas"] = 2
df2_3["replicas"] = 3

# Combinar los datos en un solo DataFrame
df2_all = pd.concat([df2_1, df2_2, df2_3], ignore_index=True)

# Cargar los archivos fase 3
file3_1 = "fase3_1replica.csv"
file3_2 = "fase3_2replica.csv"
file3_3 = "fase3_3replica.csv"

df3_1 = pd.read_csv(file3_1)
df3_2 = pd.read_csv(file3_2)
df3_3 = pd.read_csv(file3_3)

# Agregar una columna para identificar el número de réplicas
df3_1["replicas"] = 1
df3_2["replicas"] = 2
df3_3["replicas"] = 3

# Combinar los datos en un solo DataFrame
df3_all = pd.concat([df3_1, df3_2, df3_3], ignore_index=True)

# Limpiar y convertir valores
df1_all['Error %'] = df1_all['Error %'].str.replace('%',
'').astype(float)
df1_all['Throughput'] = pd.to_numeric(df1_all['Throughput'],
errors='coerce')

# Filtrar solo las filas resumen de cada archivo
df_total1 = df1_all[df1_all["Label"] == "TOTAL"].copy()
df_total1 = df_total1.sort_values(by="replicas")

df2_all['Error %'] = df2_all['Error %'].str.replace('%',
'').astype(float)
df2_all['Throughput'] = pd.to_numeric(df2_all['Throughput'],
errors='coerce')

# Filtrar solo las filas resumen de cada archivo
df_total2 = df2_all[df2_all["Label"] == "TOTAL"].copy()
df_total2 = df_total2.sort_values(by="replicas")
```

```
# Limpiar y convertir valores
df3_all['Error %'] = df3_all['Error %'].str.replace('%',
'').astype(float)
df3_all['Throughput'] = pd.to_numeric(df3_all['Throughput'],
errors='coerce')

# Filtrar solo las filas resumen de cada archivo
df_total3 = df3_all[df3_all["Label"] == "TOTAL"].copy()
df_total3 = df_total3.sort_values(by="replicas")
```

Bonus Fase 1:

Utilizamos cAdvisor y Prometheus para recopilar y visualizar métricas relacionadas con los contenedores de nuestro proyecto, entre otras estadísticas. Podemos acceder a esta herramienta por medio del puerto 8080 y 9090 respectivamente. Las gráficas y resultados se pueden apreciar de mejor manera en los documentos anexados en el repositorio de GitHub

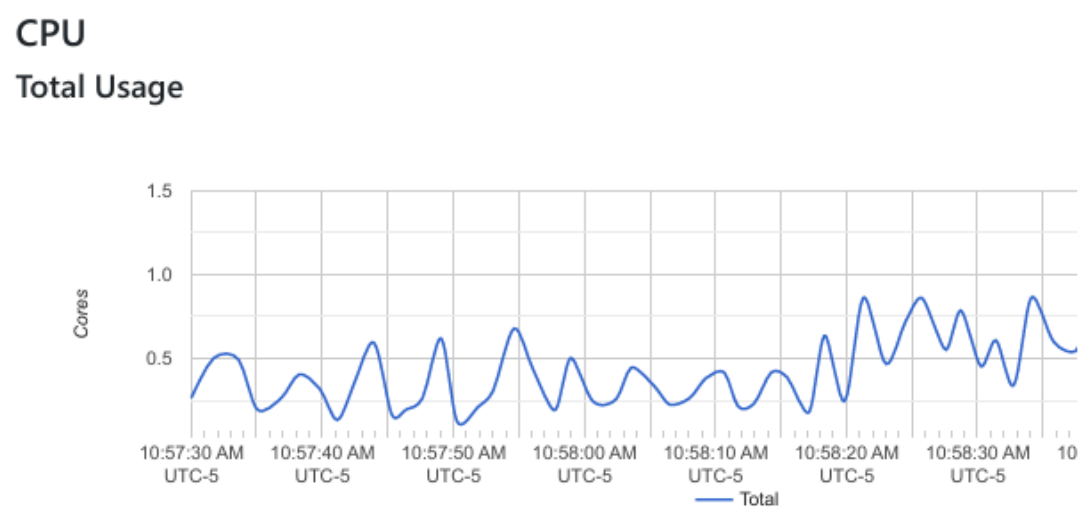
Vista previa en cAdvisor:



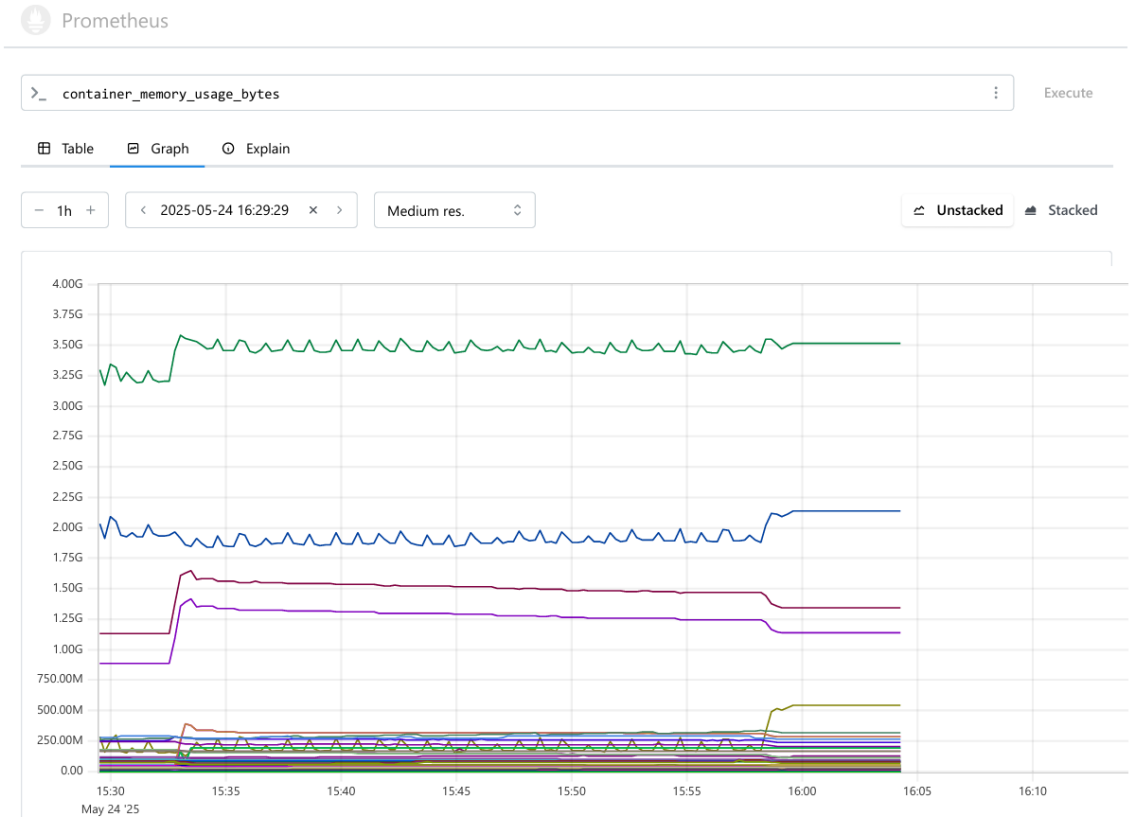
Processes

User	PID	PPID	Start Time	CPU % ▼	MEM %	
root	2,658,208	2,658,165	15:22	12.60	10.90	4
root	92	2	May19	9.70	0.00	
root	819	1	May19	5.00	7.60	2
root	16,396	16,391	May19	3.70	1.10	
root	2,663,835	1	15:27	3.30	8.00	3
student	2,011	1	May19	2.60	0.10	
root	816	1	May19	2.30	1.30	
root	820	1	May19	2.00	0.00	
student	2,671,685	2,671,609	15:32	1.90	5.80	2
root	1	0	May19	1.20	0.20	
root	3,118	815	May19	1.20	0.10	
root	1,266	1	May19	1.00	1.60	
nobody	2,658,141	2,658,120	15:22	0.70	2.30	
5050	2,673,708	2,671,687	15:33	0.70	4.40	1
lxd	2,664,597	2,664,474	15:27	0.50	3.00	1
root	831	1	May19	0.40	0.50	
student	1,792	1,525	May19	0.40	0.10	
root	16,164	1	May19	0.40	0.20	
student	2,647,853	2,647,749	14:54	0.40	0.00	
student	1,092	1	May19	0.30	0.10	
root	2,648,350	1	15:14	0.30	0.70	
root	14	2	May19	0.20	0.00	
root	828	1	May19	0.20	0.20	
root	1,391	1,363	May19	0.20	0.00	
root	2,648,521	1	15:17	0.20	0.60	

1 2 10 12



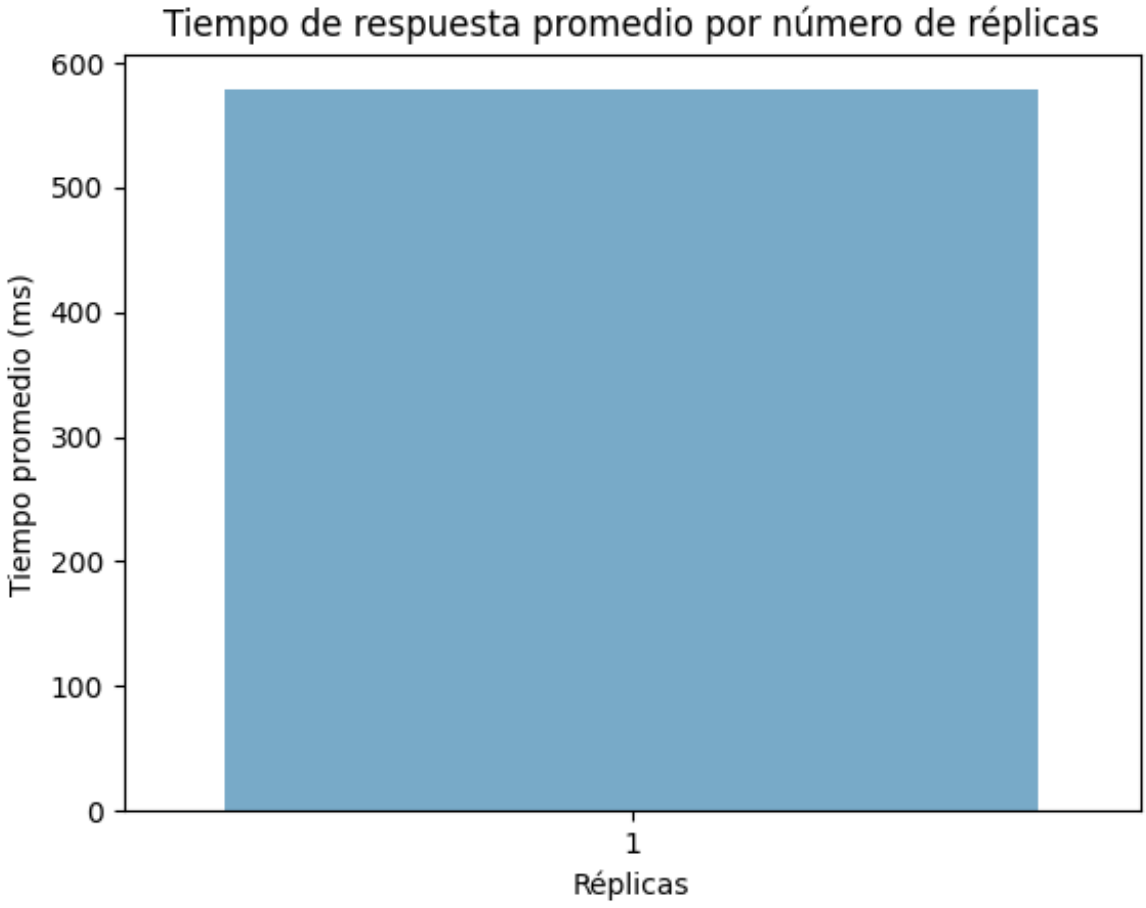
Vista previa en Prometheus:



Gráficas de la Fase 1

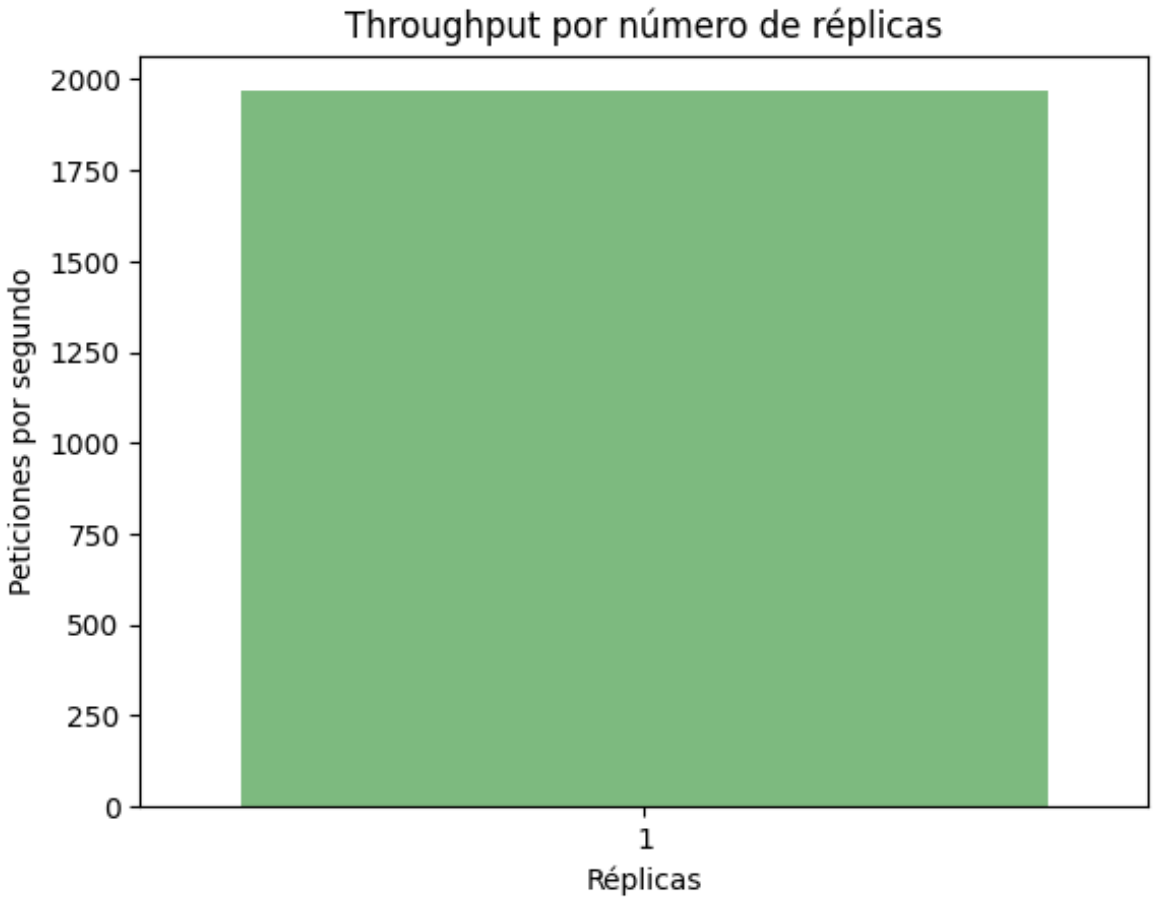
Tiempos de respuesta:

```
sns.barplot(x='replicas', y='Average', data=df_total1,
palette='Blues')
plt.title('Tiempo de respuesta promedio por número de réplicas')
plt.ylabel('Tiempo promedio (ms)')
plt.xlabel('Réplicas')
plt.show()
```



Throughput

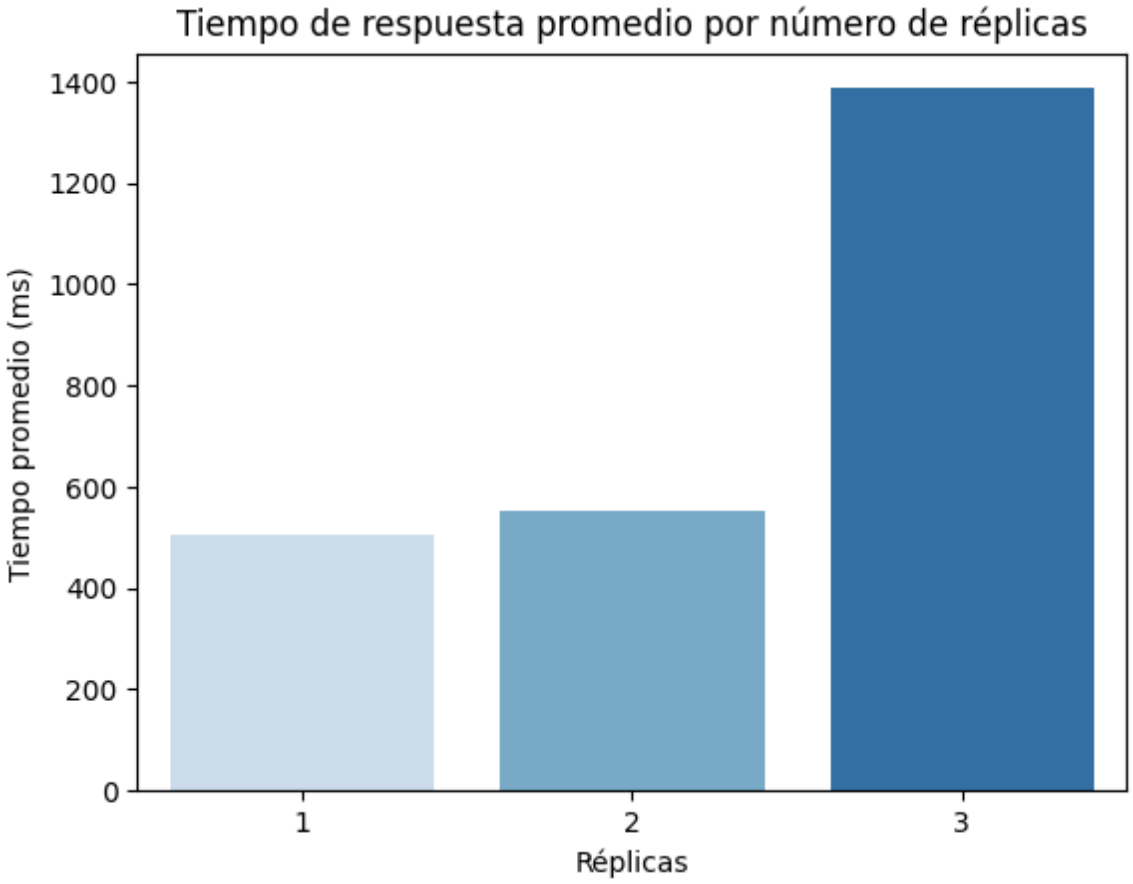
```
sns.barplot(x='replicas', y='Throughput', data=df_total1,
palette='Greens')

plt.title('Throughput por número de réplicas')

plt.ylabel('Peticiones por segundo')

plt.xlabel('Réplicas')

plt.show()
```



Gráficas de la Fase 2

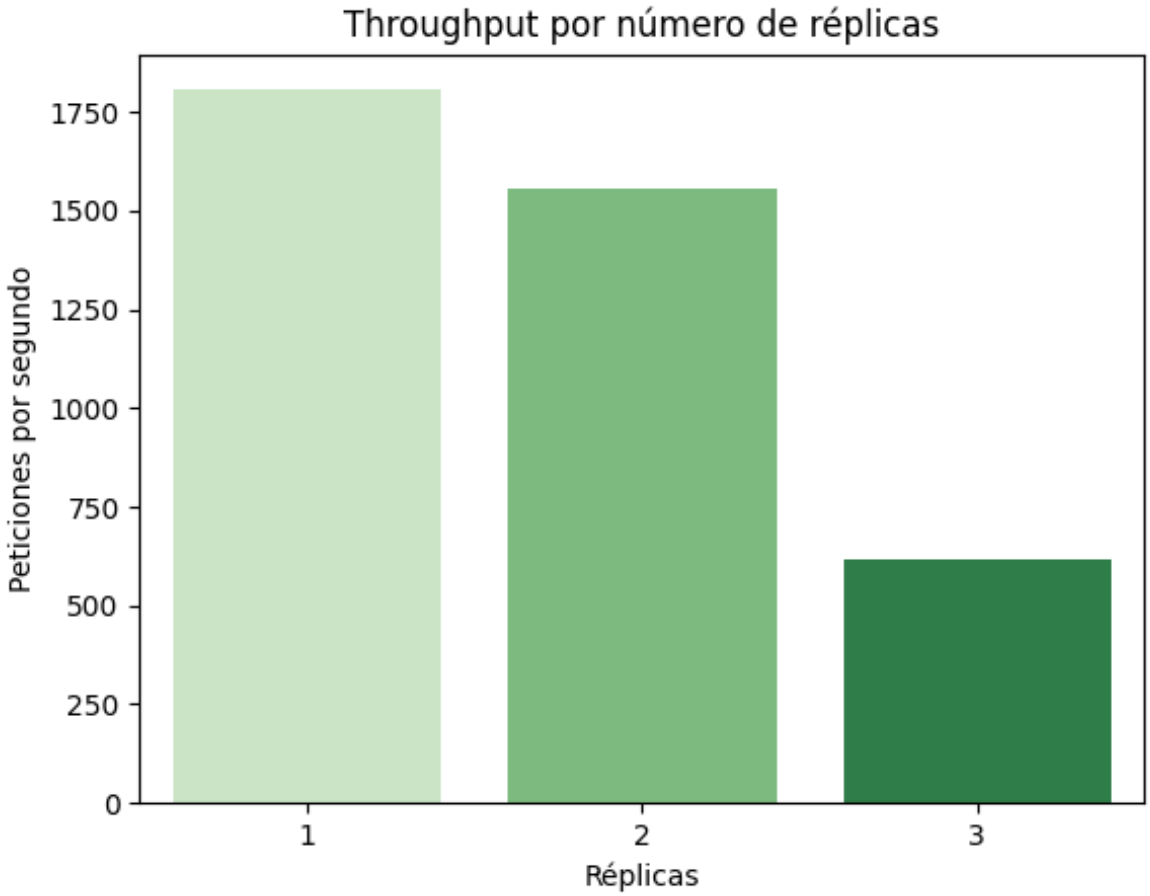
Tiempos de respuesta:

```
sns.barplot(x='replicas', y='Average', data=df_total3,
palette='Blues')
plt.title('Tiempo de respuesta promedio por número de réplicas')
plt.ylabel('Tiempo promedio (ms)')
plt.xlabel('Réplicas')
plt.show()
```



Throughput

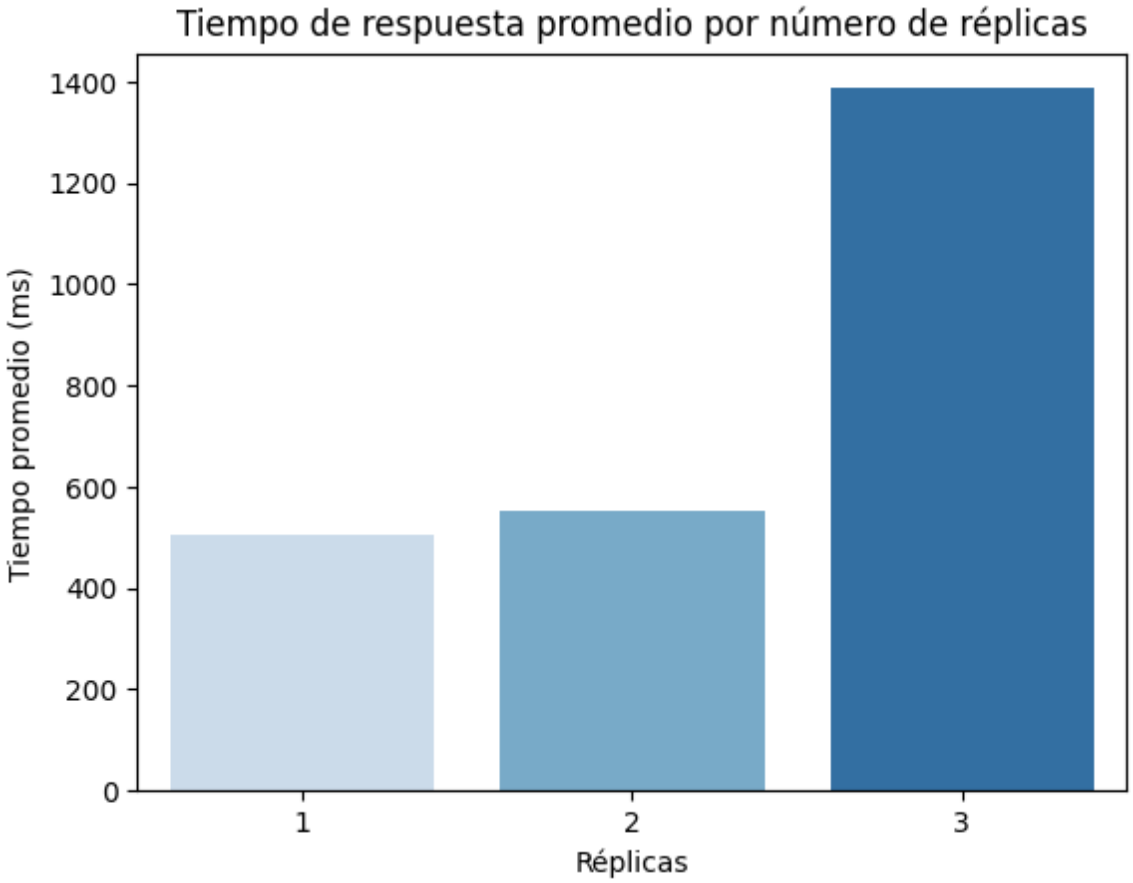
```
sns.barplot(x='replicas', y='Throughput', data=df_total3,
palette='Greens')
plt.title('Throughput por número de réplicas')
plt.ylabel('Peticiones por segundo')
plt.xlabel('Réplicas')
plt.show()
```



Gráficas de la Fase 3

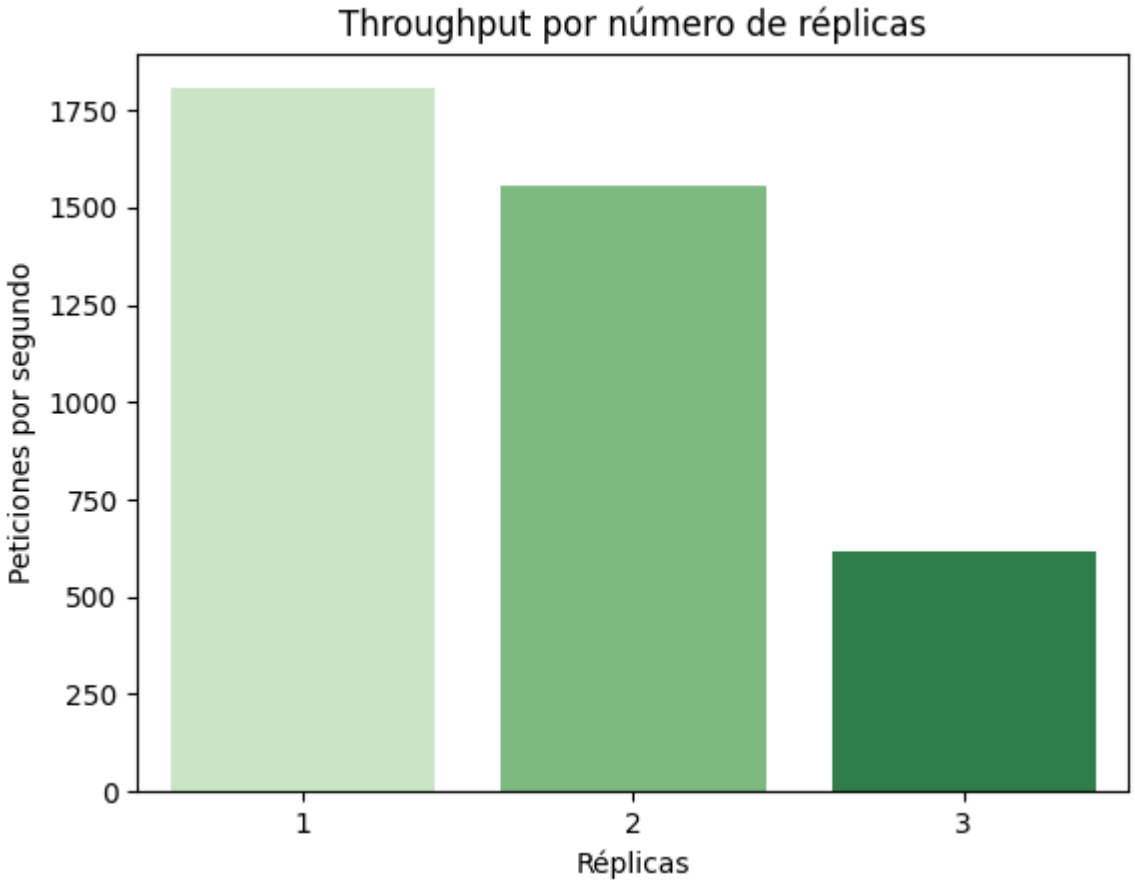
Tiempos de respuesta:

```
sns.barplot(x='replicas', y='Average', data=df_total3,
palette='Blues')
plt.title('Tiempo de respuesta promedio por número de réplicas')
plt.ylabel('Tiempo promedio (ms)')
plt.xlabel('Réplicas')
plt.show()
```



Throughput

```
sns.barplot(x='replicas', y='Throughput', data=df_total3,
palette='Greens')
plt.title('Throughput por número de réplicas')
plt.ylabel('Peticiones por segundo')
plt.xlabel('Réplicas')
plt.show()
```



Análisis

Rendimiento por Fase y Réplica

Fase	Réplicas	Avg (ms)	Std. Dev.	Error %	Throughput (req/s)	Avg. Bytes
1	-	578	1448.68	82.31%	1967.42	2806.1
2	1	571	1029.14	80.47%	2470.71	2829.6
2	2	765	2654.64	73.77%	2217.11	2940.7
2	3	923	2082.09	68.23%	1769.41	3054.8
3	1	504	2621.94	89.16%	1805.46	2669.7
3	2	551	2250.96	78.72%	1556.22	2846.5
3	3	1387	3650.16	55.10%	617.09	3286.1

Observaciones Clave

- El throughput más alto se alcanzó con 1 réplica en la fase 2.
- El porcentaje de errores fue elevado en todas las fases.
- La latencia crece en fase 3 con más réplicas.
- El tamaño promedio de respuesta también crece con réplicas.

Conclusiones

1. No hay escalabilidad lineal con réplicas.
2. El error es crítico en escenarios con más carga.
3. Fase 2 es la más eficiente.
4. Fase 3 muestra limitaciones claras bajo carga.

Recomendaciones

Para el sistema:
- Mejorar manejo de concurrencia
- Validar configuraciones de red y recursos
- Control de calidad en respuestas

Para futuras pruebas:
- Incremento gradual de carga
- Uso de métricas del sistema
- Pruebas de estrés y endurance