Pruebas de Carga

ARQUITECTURA DEL SOFTWARE

WICHAT_ES4A
UNIVERSIDAD DE OVIEDO | ESCUELA DE INGENIERÍA INFORMÁTICA

"Si no lo puedes medir, no lo puedes mejorar."

— Peter Drucker

Contenido

Informe de Resultados de Pruebas de Carga

Objetivo

Evaluar el comportamiento de la aplicación bajo condiciones de carga creciente, midiendo la escalabilidad, tiempo de respuesta y tolerancia a fallos. Se realizaron dos pruebas utilizando diferentes infraestructuras para analizar el impacto de la arquitectura en el rendimiento del sistema.

Descripción de las Infraestructuras

Infraestructura Monolítica

- **Tipo.** Monolítica, en una única instancia.
- **Instancia EC2.** *m4.xlarge* (4 vCPUs, 16GB RAM, 750 Mbps de red).
- **Componentes.** Todos los contenedores (servicios de la aplicación y *MongoDB*) alojados en la misma máquina.
- **Red.** Todo el tráfico y procesamiento se concentra en un único punto.

Infraestructura con Balanceador de Carga

- **Tipo.** Distribuida con balanceo de carga.
- **Instancias EC2.** 2 x *m4.xlarge* (servicios de aplicación) + 1 x *m4.xlarge*(MongoDB).
- **Arquitectura.**
    - Balanceador de carga distribuye peticiones entre las dos instancias de aplicación.
    - *MongoDB* alojado por separado, accesible por IP privada dentro de la misma *VPC*.

Análisis de Resultados

Primera Infraestructura

https://app.artillery.io/share/sh_2bf7231032837660a2dad4c697d4714845eb5d914e04c08879654dbb7d5bb5c4

- **Comportamiento general.** El sistema responde bien inicialmente, pero empieza a mostrar degradación grave a partir de los 6 mil usuarios virtuales (unas 600 req/s).
- **Síntoma de sobrecarga.**
  - Tiempo de respuesta (*p95*) se dispara a más de 6 segundos.
  - Errores (*vusers.failed*) aumentan drásticamente a partir del inicio de la fase *Spike*.
- **Causa probable.** Saturación de CPU, memoria y red al compartir recursos entre aplicación y base de datos en una sola instancia.

Segunda Infraestructura

https://app.artillery.io/share/sh_cc4aa62bdc5a6f47f3a727f6049f17078d5a9f33a307e48ee103393d3cf4569f

- **Mejoras observadas.**
  - El sistema soporta mayor carga antes de degradarse (unos 7 mil usuarios virtuales).
  - Los errores son más graduales y no tan abruptos como en el primer caso.
- **Síntomas persistentes.**
  - En la fase final aún aparecen picos cíclicos de latencia y errores.
  - Fallos continúan al final, indicando que aún se alcanzan los límites de la infraestructura.
- **Causa probable de mejora.**
  - Separar la base de datos de los servicios de aplicación elimina la competencia por recursos y mejora el rendimiento.
  - El balanceador de carga permite distribuir el tráfico, evitando puntos únicos de fallo.

Conclusión y Recomendaciones

Conclusión

El cambio a una arquitectura distribuida con balanceador de carga y separación de servicios ha mejorado significativamente la escalabilidad y estabilidad de la aplicación. El sistema puede manejar más usuarios, responde más rápidamente durante más tiempo y degrada de forma más controlada.

Recomendaciones

- Escalar horizontalmente aún más los servicios de aplicación mediante autoescalado, especialmente para la fase *Spike*.
- Implementar caché para reducir la presión sobre *MongoDB* en operaciones repetitivas (como preguntas para la partida).
- Configurar límite de conexiones y pool de base de datos para evitar saturación en *MongoDB*.
- Considerar usar herramientas de observabilidad como *AWS CloudWatch* para correlacionar métricas internas con los resultados de carga (también es de gran utilidad si queremos desplegar una infraestructura en la nube con autoescalado horizontal).