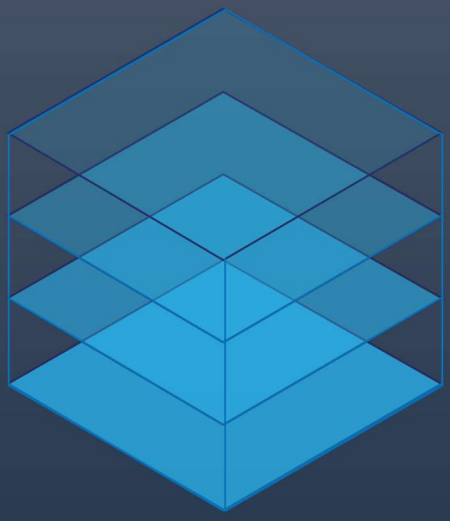
2025



MISO

Maestría en Ingeniería de Software

PROYECTO - ENTREGABLE 3
DOCUMENTO DE ESCENARIOS Y RESULTADOS DE LAS PRUEBAS DE ESTRÉS

GRUPO 11

SERGIO FERNANDO BARRERA MOLANO	202517034
HAROLD ANDRÉS BARTOLO MOSCOSO	202513889
EDWIN HERNÁN HURTADO CRUZ	202326341
JUAN JOSÉ RESTREPO BONILLA	202516633

Escenarios y resultados de las pruebas de estrés

1. Objetivo general de las pruebas.

Evaluar el comportamiento y la capacidad de la aplicación web desplegada en AWS frente a un incremento progresivo de usuarios concurrentes, identificando el punto en el que el sistema comienza a degradar su rendimiento o falla.

2. Entorno de Pruebas.

Esta sección describe la infraestructura y las condiciones bajo las cuales se realizaron las pruebas. El entorno se diseñó para replicar el comportamiento de producción en un contexto controlado, empleando recursos y servicios reales de AWS para garantizar resultados representativos.

Se tiene los siguientes componentes de arquitectura en el despliegue de AWS.

Componente	Descripción
Región AWS	us-east-1
Balanceador de carga (ALB)	Distribuye peticiones HTTP hacia las instancias EC2 de la capa web
Instancias EC2 (Auto Scaling Group)	3 instancias t3.small para la capa web
Base de datos (RDS)	PostgreSQL (db.t3.small) utilizada para autenticación y almacenamiento persistente
Almacenamiento de objetos (S3)	Contiene los archivos asociados a los usuarios
Monitoreo (CloudWatch)	Recolección de métricas de CPU, red y respuesta del balanceador
Herramienta de prueba	Apache JMeter 5.6.3 ejecutado en Windows
Tipo de prueba	Estrés (simulación progresiva de múltiples usuarios intentando autenticarse simultáneamente)

3. Escenario de prueba ejecutado.

En esta sección se describe el diseño del escenario que se utilizó para someter la aplicación a condiciones de carga extrema.

El enfoque principal fue simular múltiples usuarios concurrentes realizando la acción de inicio de sesión, lo que representa uno de los puntos más críticos de la aplicación en términos de consumo de recursos.

Parámetro	Valor
Objetivo	Evaluar el punto de falla durante autenticaciones concurrentes
Endpoint probado	POST /auth/Login GET /health GET /api/videos
Usuarios concurrentes	Incremento progresivo de 1 a 120 usuarios
Ramp-up period	60 segundos
Duración total de la prueba	2 minutos
Datos de prueba	Cuentas de usuario válidas creadas previamente

En la siguiente imagen se puede apreciar la configuración de Jmeter, donde se aprecia el Thread Group, el cual ejecuta las pruebas.

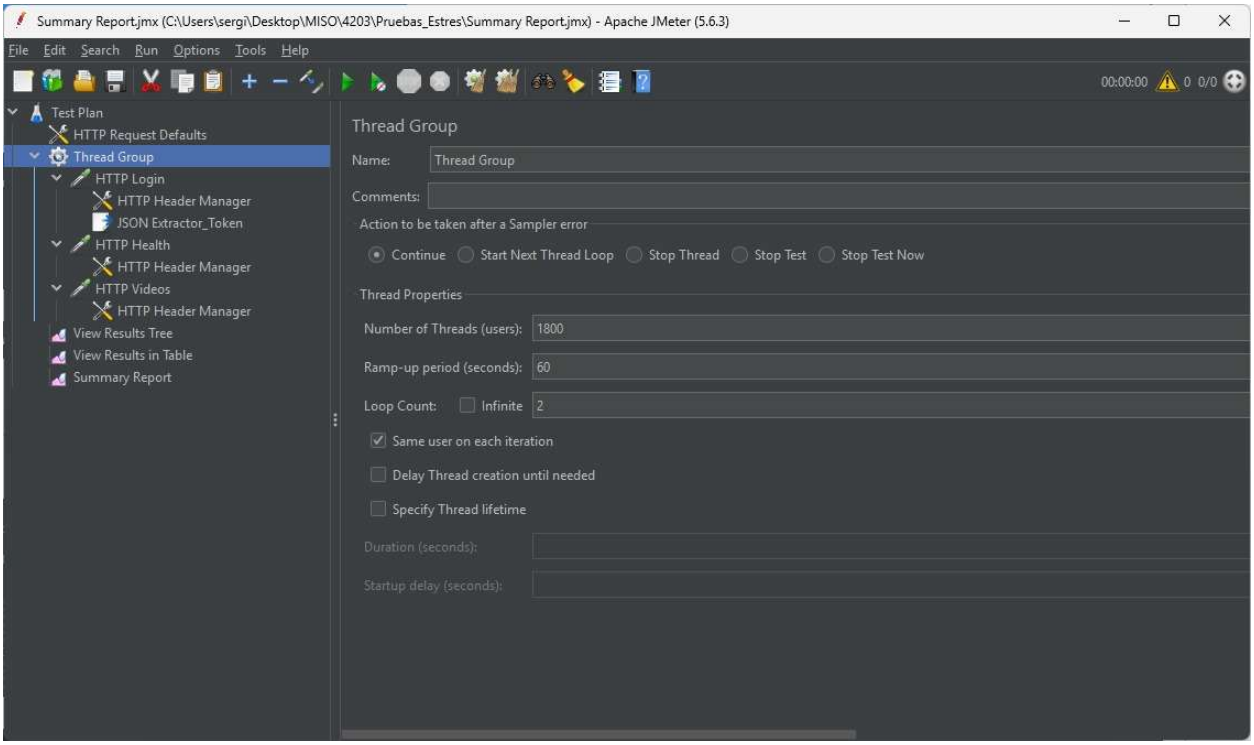


Figure 1 JMeter con la configuración cargada

En la siguiente imagen podemos observar los pasos que realiza Jmeter contra el aplicativo:

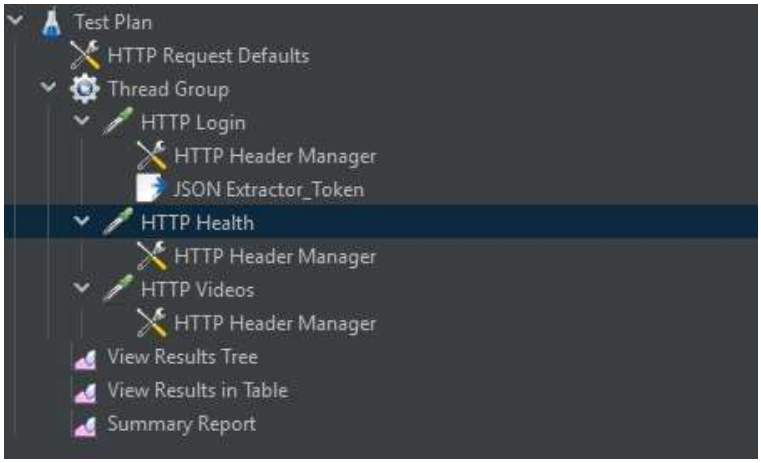


Figure 2 Thread Group que se ejecutara

4. Resultados.

A continuación, se presentan los resultados obtenidos durante la ejecución de la prueba. Se incluyen las métricas de rendimiento, los errores detectados y el comportamiento general de la aplicación ante el aumento de carga.

Al iniciar el proceso de sobrecarga de forma gradual, observamos que el aplicativo responde correctamente, sin embargo, al llegar a más de 112 usuarios concurrentes se chashea como se observa en las siguientes imágenes.

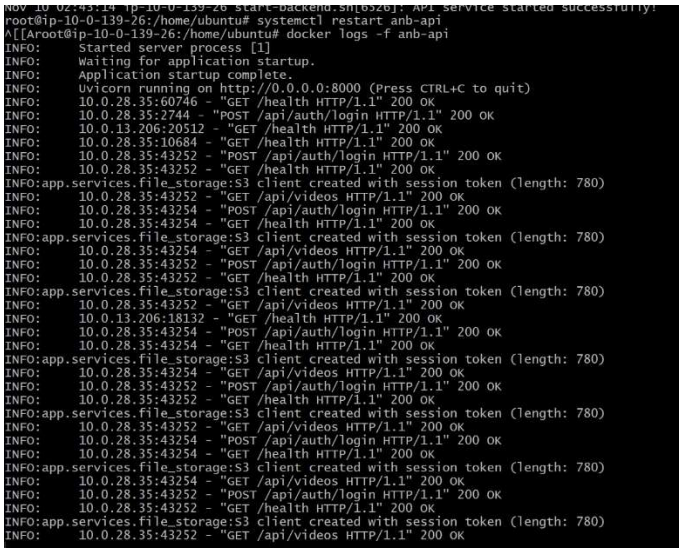


Figure 3 Aplicación bajo prueba, poco estres.

Figure 4 Aplicación bajo prueba, estres medio-alto

```
  File "/usr/local/lib/python3.12/site-packages/sqlalchemy/pool/impl.py", line 168, in _do_get
    raise exc.TimeoutError(
sqlalchemy.exc.TimeoutError: QueuePool limit of size 5 overflow 10 reached, connection timed out, timeout 30.00 (Background on this error at: https:
/sqlalche.me/e/20/3o7r)
ERROR:app.main:Unhandled exception: QueuePool limit of size 5 overflow 10 reached, connection timed out, timeout 30.00 (Background on this error at:
https://sqlalche.me/e/20/3o7r)
INFO:     10.0.28.35:39654 - "POST /api/auth/login HTTP/1.1" 500 Internal Server Error
ERROR:    Exception in ASGI application
Traceback (most recent call last):
  File "/usr/local/lib/python3.12/site-packages/uvicorn/protocols/http/httptools_impl.py", line 401, in run_asgi
    result = await app(  # type: ignore[func-returns-value]
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/site-packages/uvicorn/middleware/proxy_headers.py", line 70, in __call__
    return await self.app(scope, receive, send)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/site-packages/fastapi/applications.py", line 1133, in __call__
    await super().__call__(scope, receive, send)
  File "/usr/local/lib/python3.12/site-packages/starlette/applications.py", line 113, in __call__
    await self.middleware_stack(scope, receive, send)
  File "/usr/local/lib/python3.12/site-packages/starlette/middleware/errors.py", line 186, in __call__
    raise exc
  File "/usr/local/lib/python3.12/site-packages/starlette/middleware/errors.py", line 164, in __call__
    await self.app(scope, receive, _send)
  File "/usr/local/lib/python3.12/site-packages/starlette/middleware/cors.py", line 85, in __call__
    await self.app(scope, receive, send)
  File "/usr/local/lib/python3.12/site-packages/starlette/middleware/exceptions.py", line 63, in __call__
    await wrap_app_handling_exceptions(self.app, conn)(scope, receive, send)
  File "/usr/local/lib/python3.12/site-packages/starlette/_exception_handler.py", line 53, in wrapped_app
    raise exc
  File "/usr/local/lib/python3.12/site-packages/starlette/_exception_handler.py", line 42, in wrapped_app
    await app(scope, receive, sender)
  File "/usr/local/lib/python3.12/site-packages/fastapi/middleware/asyncexitstack.py", line 18, in __call__
    await self.app(scope, receive, send)
  File "/usr/local/lib/python3.12/site-packages/starlette/routing.py", line 716, in __call__
    await self.middleware_stack(scope, receive, send)
  File "/usr/local/lib/python3.12/site-packages/starlette/routing.py", line 736, in app
    await route.handle(scope, receive, send)
  File "/usr/local/lib/python3.12/site-packages/starlette/routing.py", line 290, in handle
    await self.app(scope, receive, send)
  File "/usr/local/lib/python3.12/site-packages/fastapi/routing.py", line 123, in app
    await wrap_app_handling_exceptions(app, request)(scope, receive, send)
  File "/usr/local/lib/python3.12/site-packages/starlette/_exception_handler.py", line 53, in wrapped_app
    raise exc
```

Figure 5 Aplicación completamente crasheada - No permite más ingresos

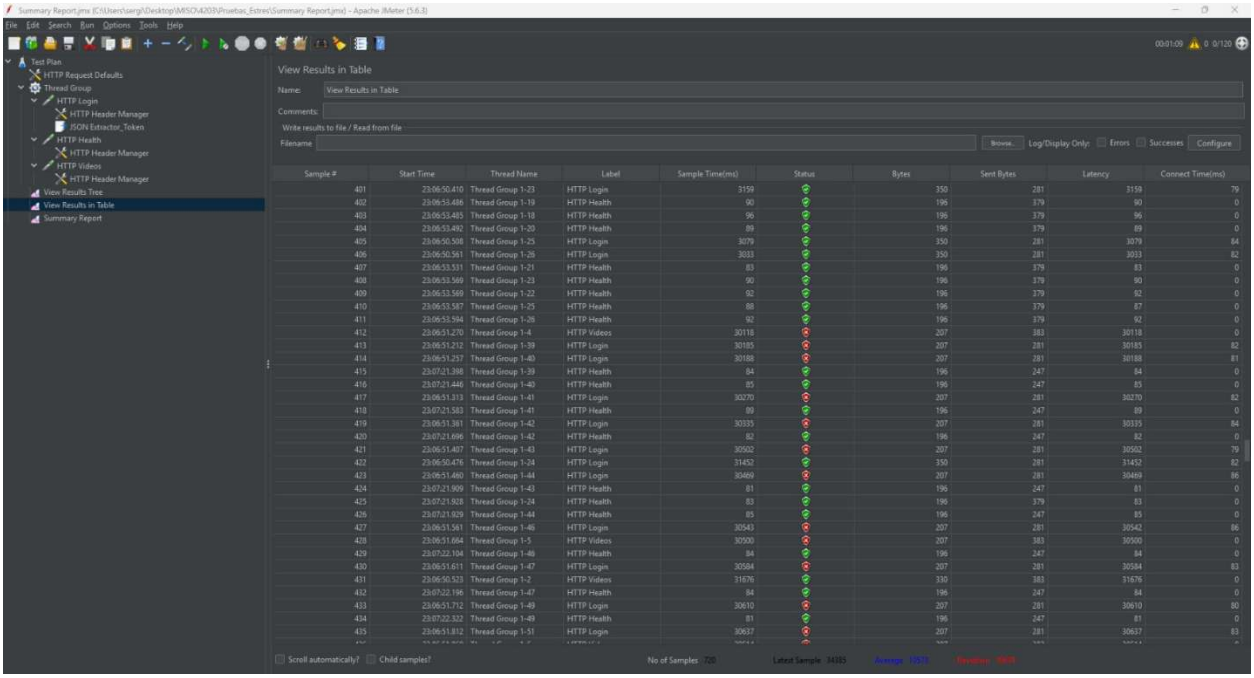


Figure 6 JMeter en punto de quiebre, estres medio alto

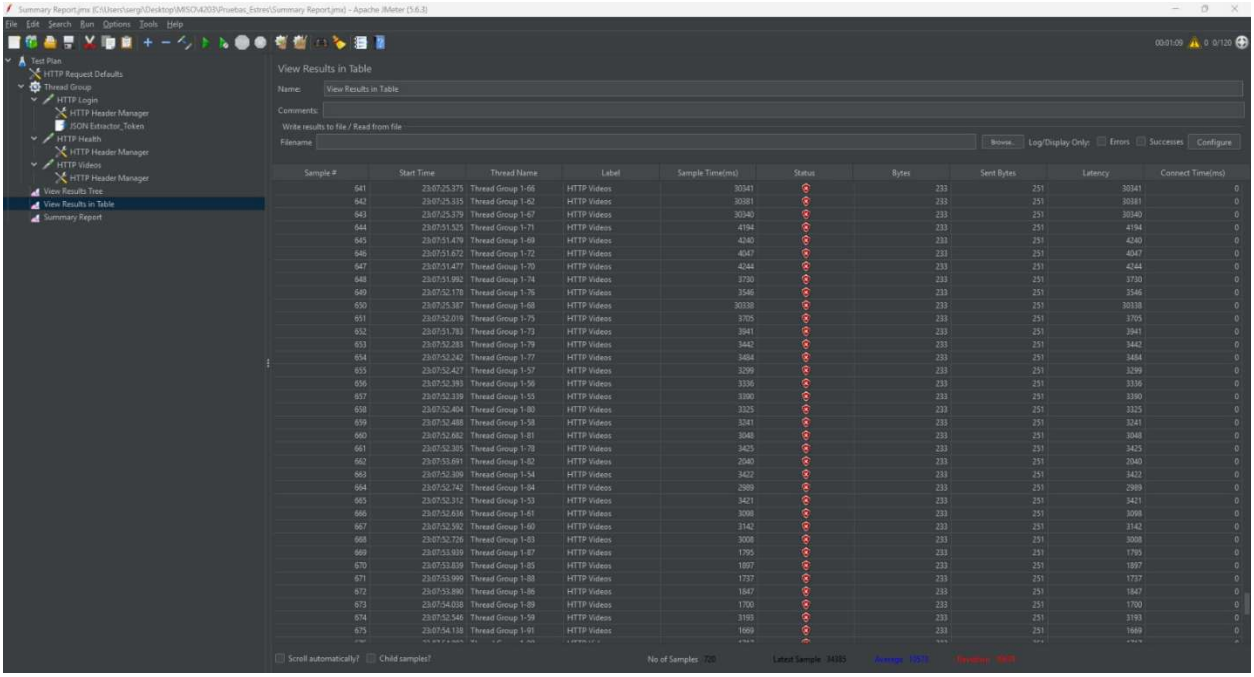


Figure 7 JMeter cuando el aplicación está completamente crasheado

Table 1 Análisis de los datos obtenidos de JMeter

Petición	Promedio (ms)	Error %	Throughput (req/seg)	Observaciones
HTTP Login	20,830	33.33%	1.4	El tiempo promedio de respuesta fue significativamente alto, evidenciando saturación en el proceso de autenticación. Un tercio de las solicitudes fallaron, indicando un posible cuello de botella en la conexión con la base de datos o en el servicio de autenticación.
HTTP Health	110	0.00%	1.4	Este endpoint, de baja complejidad, se mantuvo estable y rápido durante toda la prueba, lo que sugiere que el entorno general y el balanceador de

				carga estaban funcionando correctamente.
HTTP Videos	10,779	42.92%	1.4	Se presentaron tiempos de respuesta altos y una tasa de error superior al 40%. Esto indica que las peticiones a endpoints que requieren autenticación y acceso a datos presentan serios problemas de concurrencia.

En promedio, el tiempo de respuesta global fue de 10,573 ms (≈10,5 segundos), con un porcentaje de error total del 25.42% y un throughput combinado de 4.1 solicitudes por segundo.

Estos valores reflejan que la aplicación no mantiene estabilidad bajo carga moderada, y que el componente de autenticación (/auth/login) es el más sensible al estrés.
El alto *Standard Deviation* (>19.000 ms) evidencia una respuesta inconsistente: algunos usuarios obtuvieron respuesta en menos de un segundo, mientras otros esperaron más de 60 segundos o recibieron error.

5. Conclusiones.

La aplicación presenta limitaciones de concurrencia: El sistema mantiene estabilidad hasta aproximadamente 100-110 usuarios simultáneos; a partir de ese punto, los tiempos de respuesta superan los 20 segundos y aumentan los errores HTTP 500, evidenciando saturación en el backend.

El proceso de autenticación es el principal cuello de botella: El endpoint /auth/login concentra el mayor tiempo de respuesta y más del 30% de errores, lo que sugiere sobrecarga en la gestión de sesiones o conexiones a la base de datos.

Se requieren ajustes de escalabilidad y optimización: Es necesario revisar las políticas de Auto Scaling y optimizar el uso de recursos (pool de conexiones, caché y uso eficiente de RDS) para mejorar la respuesta del sistema bajo carga creciente.

6. Análisis Técnico de la Falla.

Cuello de botella en autenticación: El endpoint /auth/login muestra el mayor tiempo de respuesta y un tercio de los errores. Esto sugiere saturación en la gestión de sesiones o consultas al RDS.

Concurrencia limitada: La aplicación mantiene estabilidad hasta un número bajo de usuarios simultáneos. El throughput estable de 1.4 req/s por endpoint indica que los recursos del backend están llegando rápidamente a su límite de procesamiento.

Auto Scaling sin respuesta: Aunque se esperaba que el grupo de Auto Scaling reaccionara ante el incremento de carga, no se observó evidencia de nuevas instancias desplegadas durante la prueba, lo que sugiere una política de escalado toma tiempo, más cuando se tiene una prueba de estrés tipo spike, que envía demasiados usuarios de forma muy rápida, por tanto, se debe planificar aumentar las EC2 que se tiene en IDLE para este tipo de peticiones.

Diferencia entre endpoints: El endpoint /health responde correctamente, lo que demuestra que la infraestructura y el balanceador funcionan. El problema radica en los servicios que acceden a datos (login y videos), ya que si se cae el Login, no se podría acceder a la aplicación ya que no pasaría la validación de token para acceder a los demás APIs.