

Documentación del Proyecto – Unidades 2 y 3

Administración de Redes y Sistemas Computacionales

Diego Salvador

Julio 2025

1. Descripción General

Este proyecto consiste en el desarrollo de una **aplicación de gestión de tareas colaborativas** basada en una arquitectura de microservicios, desplegada con Docker y orquestada mediante Docker Compose. La aplicación se asegura mediante técnicas de cifrado SSL/TLS, protección contra ataques DoS y mecanismos de alta disponibilidad.

2. Unidad 2: Construcción y Orquestación de Microservicios

2.1. Arquitectura General

- **user-service**: gestión de usuarios (registro, consulta).
- **task-service**: gestión de tareas (creación, actualización, consulta).
- **API Gateway (Nginx Proxy Manager)**: enruta el tráfico de forma centralizada.

2.2. user-service

- **Lenguaje**: Node.js (Express)
- **Base de datos**: PostgreSQL
- **Endpoints**:
 - POST /users
 - GET /users
 - GET /users/:id
 - GET /health
- **Validaciones**: email único, campos obligatorios.

2.3. task-service

- **Lenguaje**: Node.js (Express)
- **Base de datos**: PostgreSQL
- **Comunicación interna**: consulta a user-service para validar userId.
- **Endpoints**:
 - POST /tasks
 - GET /tasks
 - GET /tasks/:id
 - PUT /tasks/:id
 - GET /tasks?user_id=X

- GET /health
- **Estados de tareas:** pendiente, en progreso, completada.

2.4. Orquestación con Docker Compose

- Archivo `docker-compose.yml`:
 - Define `user-service`, `task-service` y `nginx-proxy-manager`.
 - Redes personalizadas para comunicación segura.
 - Volúmenes persistentes para bases de datos y configuración.
 - Healthchecks para todos los servicios.

2.5. Rutas configuradas

- `/api/users/*` → `user-service`
- `/api/tasks/*` → `task-service`
- Subdominio `admin.` → Interfaz NPM

3. Unidad 3: Seguridad y Alta Disponibilidad

3.1. API Gateway Seguro con Certificados SSL/TLS

Para asegurar la comunicación segura entre los usuarios y la aplicación, se implementó un proxy inverso mediante **Nginx Proxy Manager (NPM)**. Este actúa como punto central de entrada y permite habilitar HTTPS con certificados SSL válidos emitidos automáticamente por Let's Encrypt.

Pasos Realizados

1. Registro del dominio y configuración DNS:

- Se utilizó el dominio `proyectoadmin.mo00.com`, registrado en `https://freedns.afraid.org`.
- El dominio se configuró para apuntar a la dirección IP pública de la máquina virtual (VM) donde corre Docker y Nginx Proxy Manager.

2. Acceso a Nginx Proxy Manager:

- Se accedió a la interfaz web de NPM mediante el puerto 81, por ejemplo: `http://proyectoadmin.mo00.com:81`.
- Desde allí, se configuran todos los redireccionamientos necesarios.

3. Configuración del primer proxy host (API):

- **Domain name:** proyectoadmin.mooo.com
- **Forward hostname:** localhost
- **Forward port:** 81
- **SSL:**
 - Certificado SSL generado con Let's Encrypt.
 - Force SSL activado para redireccionar todo el tráfico a HTTPS.
- **Advanced:**

```
location /api/users/ {
    proxy_pass http://user-service:3000/users/;
}
location /api/tasks/ {
    proxy_pass http://task-service:3000/tasks/;
}
```

4. Configuración del segundo proxy host (admin):

- **Domain name:** admin.proyectoadmin.mooo.com
- **Forward hostname:** localhost
- **Forward port:** 81
- **Custom location:** /admin
- **SSL:** Certificado generado y Force SSL habilitado.

Opciones de Seguridad en SSL

En la pestaña SSL de NPM se configuraron las siguientes opciones:

- **Force SSL:** redirige automáticamente todas las peticiones HTTP a HTTPS.
- **HTTP/2 Support:** habilita el protocolo HTTP/2 para mejorar el rendimiento de carga.
- **HSTS Enabled (HTTP Strict Transport Security):** obliga a los navegadores a usar solo HTTPS.
- **HSTS Subdomains:** aplica HSTS también a todos los subdominios del dominio principal.
- **Block Common Exploits:** activa protecciones básicas contra ataques comunes.

Configuración de TLS 1.2+

TLS (Transport Layer Security) es el protocolo que cifra las conexiones HTTPS.

- **TLS 1.2**: estándar seguro y ampliamente usado.
- **TLS 1.3**: versión más reciente y aún más segura.

Al generar un certificado con Let's Encrypt mediante NPM y activar las opciones de seguridad mencionadas, el sistema ya está usando por defecto **TLS 1.2** y **TLS 1.3**, cumpliendo con los estándares actuales de seguridad sin necesidad de configuración adicional.

Test de seguridad online

Para validar externamente la configuración de seguridad HTTPS del dominio, se utilizó la herramienta gratuita **SSL Labs Test** (<https://www.ssllabs.com/ssltest/>). Esta permite verificar que el servidor utiliza versiones modernas de TLS (como TLS 1.2 y 1.3), certificados válidos, algoritmos de cifrado seguros y que no existen vulnerabilidades conocidas. Además, entrega una calificación de seguridad (como A o A+), lo cual es útil para confirmar que la conexión está correctamente protegida.

Evidencia

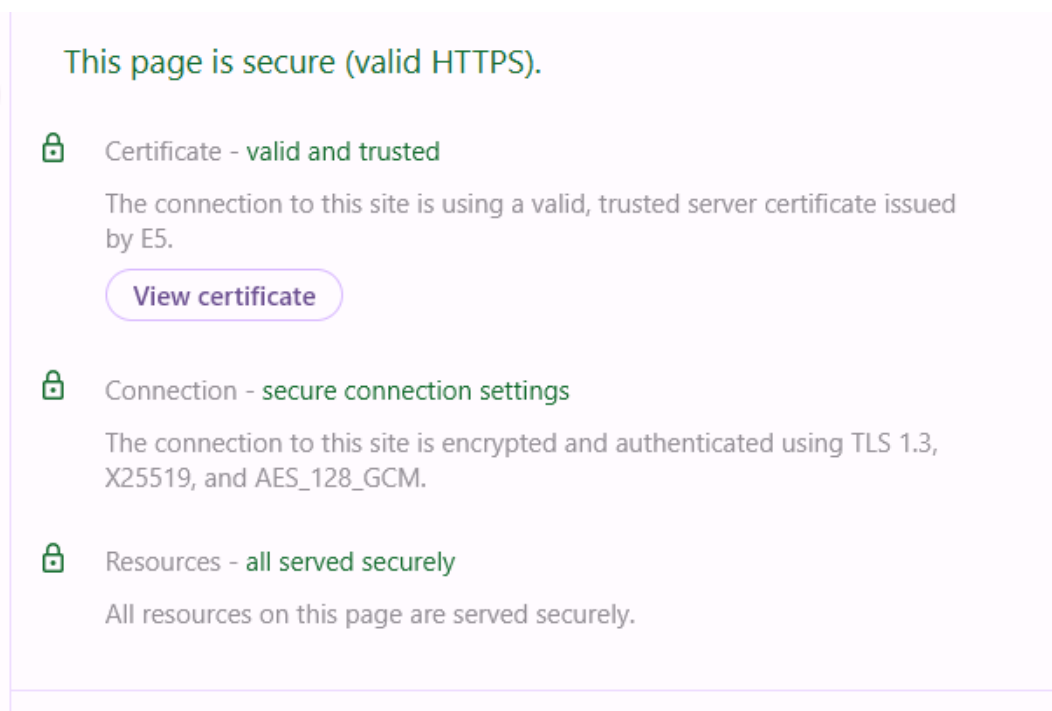


Figura 1: Conexión segura usando HTTPS y TLS 1.3



Figura 2: Certificado SSL válido emitido por Let's Encrypt

3.2. Análisis y Mitigación de Ataques DoS

Sistema de monitoreo y recolección de logs

Con el objetivo de observar y analizar el comportamiento del sistema ante ataques de denegación de servicio (DoS), se implementó una arquitectura de monitoreo compuesta por tres partes:

- **Máquina objetivo (donde residen los microservicios):** Se diseñó un `docker-compose` que despliega tres contenedores clave:
 - `node-exporter`, para exponer métricas del sistema operativo (CPU, memoria, disco, red).
 - `cAdvisor`, para recolectar métricas de uso por contenedor Docker.
 - `Filebeat`, encargado de recolectar logs tanto de los contenedores como del proxy inverso Nginx Proxy Manager.

docker-compose:

version: '3.8'

services:

cadvisor:

image: gcr.io/cadvisor/cadvisor:v0.47.2

container_name: cadvisor

network_mode: host

command:

- '--housekeeping_interval=10s'
- '--docker_only'

volumes:

- /:/rootfs:ro
- /var/run:/var/run:ro
- /sys:/sys:ro
- /var/lib/docker:/var/lib/docker:ro

restart: unless-stopped

filebeat:

image: docker.elastic.co/beats/filebeat:8.8.0

container_name: filebeat

user: root

volumes:

- /var/lib/docker/containers:/var/lib/docker/containers:ro
- /var/run/docker.sock:/var/run/docker.sock
- ./filebeat/filebeat.yml:/usr/share/filebeat/filebeat.yml:ro
- ../ProyectoAdmin/data/npm/logs:/usr/share/filebeat/npm-logs:ro

restart: unless-stopped

node-exporter:

image: prom/node-exporter:latest

container_name: node-exporter

ports:

- "9100:9100"

pid: "host"

volumes:

- /proc:/host/proc:ro
- /sys:/host/sys:ro
- /:/rootfs:ro

command:

- '--path.procfs=/host/proc'
- '--path.sysfs=/host/sys'
- '--path.rootfs=/rootfs'

restart: unless-stopped

Archivo de configuración de filebeat para recolección específica de logs:

```
filebeat.inputs:
  # Logs de contenedores Docker
  - type: container
    paths:
      - /var/lib/docker/containers/*//*.log
    json.keys_under_root: true
    json.add_error_key: true
    processors:
      - add_docker_metadata: ~
      - drop_event:
          when.or:
            - contains:
                container.name: "cadvisor"
            - contains:
                container.name: "filebeat"

  # Logs del Nginx Proxy Manager
  - type: log
    paths:
      - /usr/share/filebeat/npm-logs/*_access.log
      - /usr/share/filebeat/npm-logs/*_error.log
    multiline.pattern: '^\d{1,3}\.'
    multiline.negate: true
    multiline.match: after
    fields:
      service: nginx-proxy
    processors:
      - add_fields:
          target: ''
          fields:
            service_type: reverse-proxy

output.elasticsearch:
  hosts: ["http://34.0.50.155:3100"]
  index: "filebeat-docker-%{+yyyy.MM.dd}"

setup.ilm.enabled: false
setup.template.name: "filebeat-docker"
setup.template.pattern: "filebeat-docker-*
```

- **Máquina de monitoreo:** Se desplegó un entorno de análisis con:

- Prometheus para almacenamiento de métricas.
- Grafana para la visualización de datos de rendimiento en tiempo real.
- Elasticsearch y Kibana para el almacenamiento y análisis visual de logs.

El monitoreo busca observar en tiempo real el efecto del ataque sobre los contenedores (CPU, red, memoria), y registrar evidencia desde los logs del proxy y contenedores. Los registros del proxy pueden permitir identificar las rutas atacadas, errores frecuentes (como 503 y 429) y direcciones IP involucradas. Se busca evaluar el impacto de las medidas de mitigación, a través de comparaciones directas.

- **Máquina de ataque y análisis de pruebas de estrés:** Como parte de la infraestructura, se incorpora una máquina dedicada para la ejecución de pruebas de carga y simulación de ataques de denegación de servicio (DoS). Desde esta máquina se utiliza la herramienta **ab** (*Apache Benchmark*) para enviar solicitudes intensivas a los endpoints expuestos mediante el proxy inverso Nginx Proxy Manager.

Esta máquina permite:

- Generar tráfico masivo con distintos niveles de concurrencia y volumen total de peticiones.
- Medir la capacidad de respuesta del sistema ante condiciones de carga extrema.
- Obtener métricas de latencia, tasa de errores y throughput.

La salida de **ab** se procesa utilizando Python, empleando bibliotecas como **pandas** y **matplotlib** para generar visualizaciones que permiten analizar:

- Tiempo promedio de respuesta por solicitud.
- Porcentaje de errores HTTP (como 503, 502).
- Tasa de peticiones completadas por segundo.

Este componente complementa el sistema de monitoreo, entregando una base cuantitativa adicional para contrastar el comportamiento del sistema antes y después de aplicar medidas de mitigación.

Ejecución de pruebas de carga antes de implementar medidas

Prueba	Requests	Concurrency	Requests/s	Time/request (ms)	Errores
1	100	5	76	65.8	0
2	1000	50	224	222.7	0
3	5000	200	234	851.2	0
4	10000	500	219	2279.8	0
5	20000	1000	198	5039.3	5
6	40000	1200	213	5630	53
7	40000	2000	247	8068	29908

Cuadro 1: Resultados de pruebas de carga con **ab**

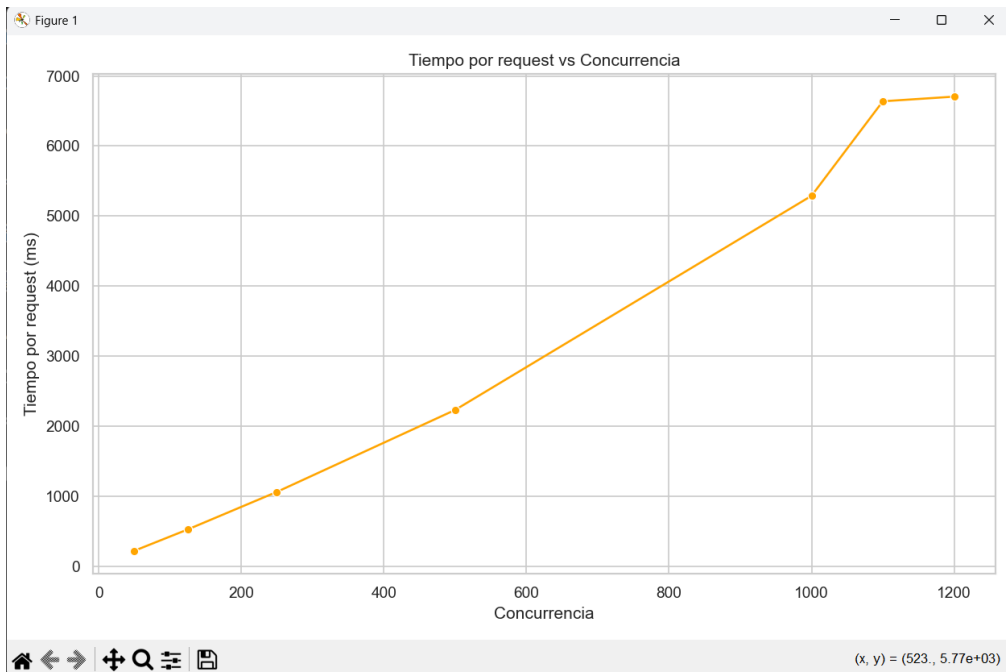


Figura 3: Tiempo por request v/s concurrencia

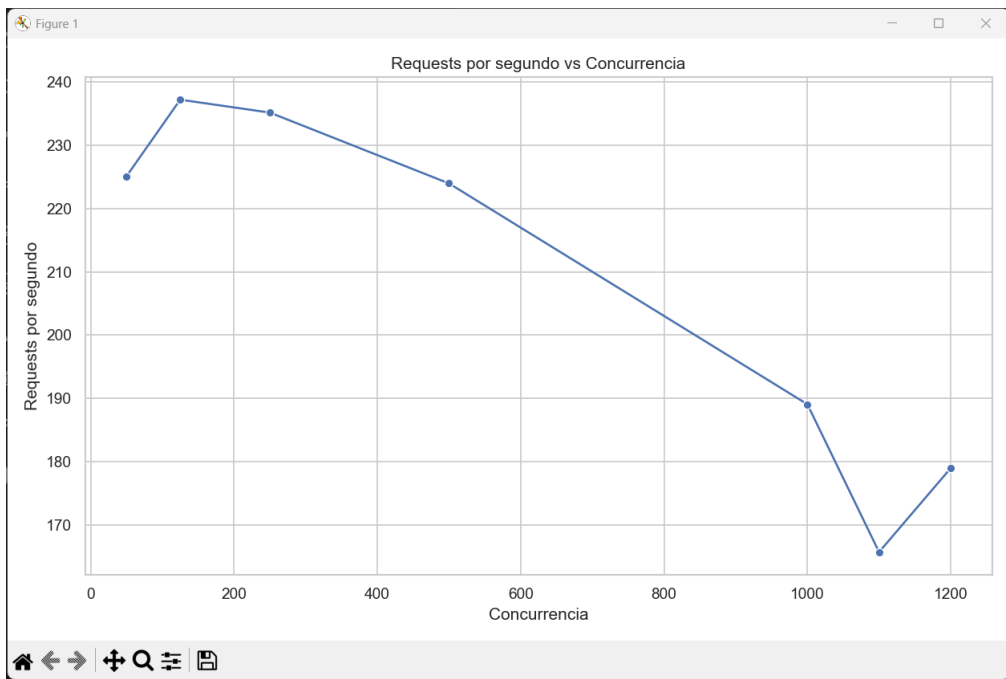


Figura 4: Requests/s v/s concurrencia

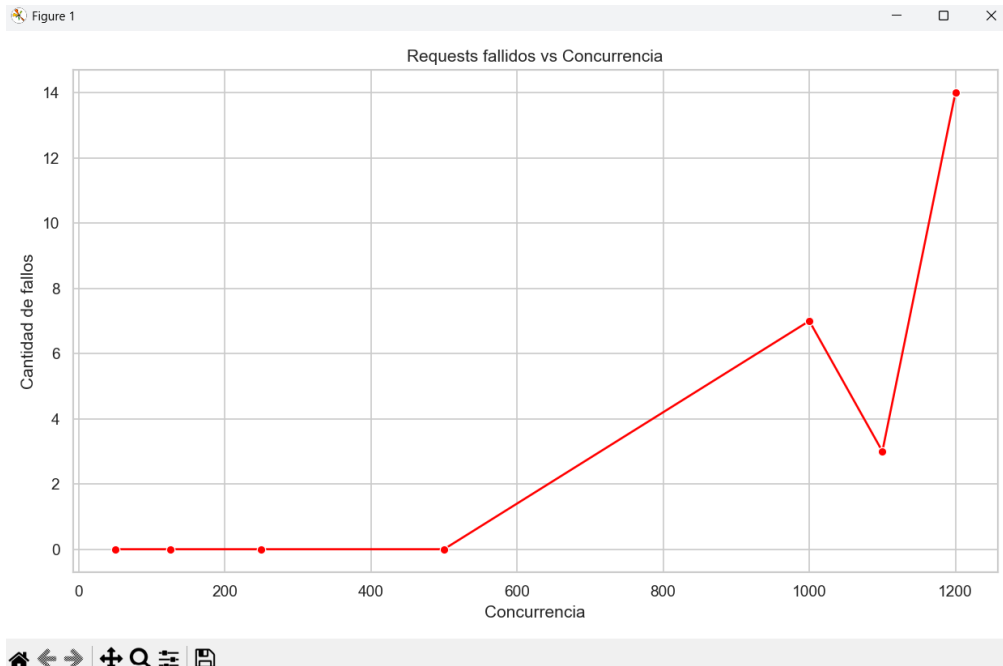


Figura 5: Requests fallidos v/s concurrencia

Problemas encontrados al aumentar la concurrencia

```
SSL handshake failed (1).
005E248EF87D0000:error:0A000126:SSL routines:ssl3_read_n:unexpected eof while reading:../ssl/record/rec_layer_s3.c:317:
SSL handshake failed (1).
005E248EF87D0000:error:0A000126:SSL routines:ssl3_read_n:unexpected eof while reading:../ssl/record/rec_layer_s3.c:317:
SSL handshake failed (1).
005E248EF87D0000:error:0A000126:SSL routines:ssl3_read_n:unexpected eof while reading:../ssl/record/rec_layer_s3.c:317:
```

Figura 6: SSL handshake failed

El error “SSH handshake failed” ocurre cuando el cliente no puede establecer una conexión segura con el servidor mediante SSH. Esto pasa porque el servidor está tan saturado por el ataque de denegación de servicio (DoS) que no tiene recursos suficientes para responder. Al recibir demasiadas solicitudes al mismo tiempo, el sistema se queda sin capacidad para manejar nuevas conexiones, incluyendo las de SSH, y por eso falla el proceso inicial de conexión.

```
worker_connections are not enough, reusing
connections 2025/07/02 05:48:23 [warn] 207#207: 512
worker_connections are not enough, reusing
connections 2025/07/02 05:48:24 [warn] 206#206: 512
```

Figura 7: worker_connections are not enough

Un worker en Nginx es un proceso que se encarga de atender las conexiones entrantes. Cada uno puede manejar muchas conexiones, pero solo hasta el límite definido por wor-

ker_connections. Cuando en los logs de Nginx Proxy Manager aparece el mensaje “worker_connections are not enough, reusing connections” durante un ataque DoS, significa que ese límite se alcanzó y Nginx ya no puede abrir nuevas conexiones. En su lugar, intenta reutilizar conexiones existentes para seguir funcionando, lo cual puede causar errores o lentitud. Este mensaje indica que el servidor está sobrecargado por la cantidad de solicitudes simultáneas y necesita ser ajustado para soportar mayor concurrencia.

```
[02/Jul/2025:06:16:46 +0000] - - 499 - GET https
proyectoadmin.mo00.com "/api/users/" [Client
181.163.86.126] [Length 0] [Gzip -] [Sent-to
localhost] "ApacheBench/2.3" "-"
```

Figura 8: Error 499

El código 499 en los logs de Nginx significa que el cliente cerró la conexión antes de que el servidor pudiera responder. En este caso, el cliente es la herramienta ApacheBench, que al lanzar muchas peticiones durante el ataque DoS, puede interrumpir conexiones si no recibe respuesta rápida. Esto ocurre porque el servidor está saturado o responde demasiado lento, así que ApacheBench simplemente cancela la solicitud antes de tiempo. El error 499 es una clara señal de que el servidor no alcanza a procesar todas las peticiones debido a la carga extrema.

Carga del sistema

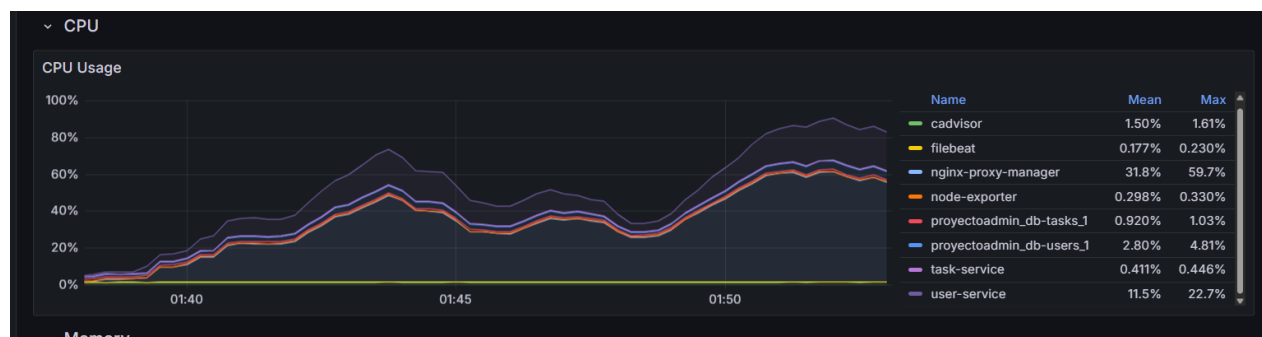


Figura 9: Gráfico de uso de CPU por contenedor en concurrencia máxima

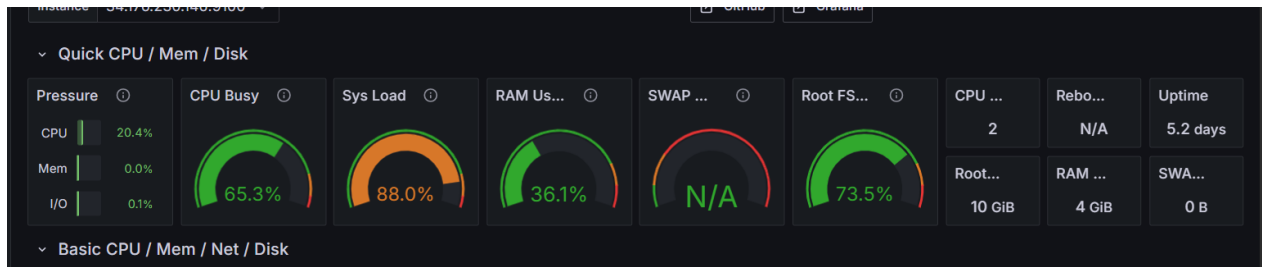


Figura 10: Carga máxima del sistema

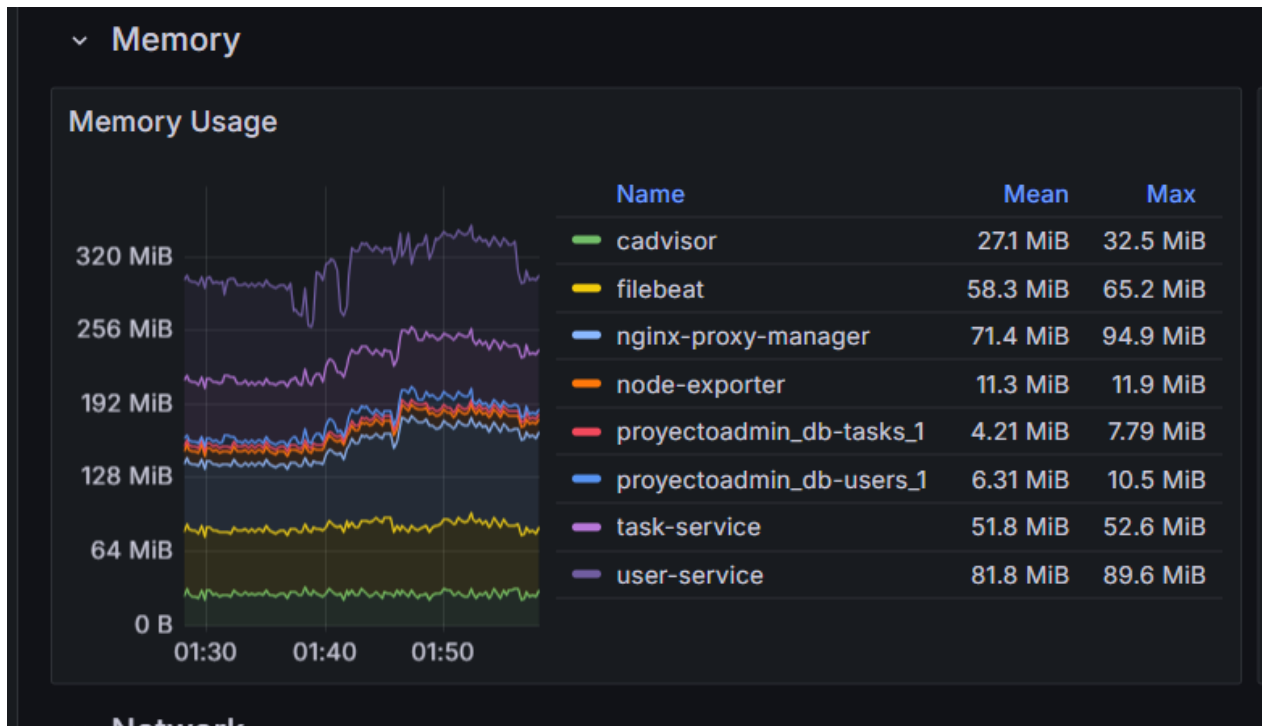


Figura 11: Memoria por contenedor en carga máxima del sistema

Medidas implementadas

1. Definición de zonas de protección global en nginx.conf:

- Se modificó el archivo `nginx.conf` utilizado por Nginx Proxy Manager, montado externamente con un volumen para persistencia.
- Dentro del bloque `http{}`, se definieron zonas compartidas de memoria para limitar peticiones y conexiones:

```
limit_req_zone $binary_remote_addr zone=mylimit:10m rate=5r/s;
limit_conn_zone $binary_remote_addr zone=addr:10m;
```

- `limit_req_zone`: crea una zona de 10MB para limitar la frecuencia de peticiones por dirección IP, permitiendo 5 solicitudes por segundo.
- `limit_conn_zone`: permite contar cuántas conexiones simultáneas mantiene cada IP, usando también 10MB.

2. Aplicación de reglas por Proxy Host en NPM:

- Desde la interfaz gráfica de NPM, se editaron manualmente los bloques `location /api/users/` y `/api/tasks/` en la pestaña **Advanced**.
- Se añadieron las siguientes reglas para aplicar los límites definidos:

```
limit_req zone=mylimit burst=10 nodelay;
limit_conn addr 10;
```

- `limit_req`: aplica la zona `mylimit` y permite ráfagas (burst) de hasta 10 solicitudes sin demora artificial.
- `limit_conn`: restringe a un máximo de 10 conexiones simultáneas por IP usando la zona `addr`.

3. Refuerzo con timeouts de conexión:

- También en el bloque `http{}`, se añadieron restricciones de tiempo para evitar ataques tipo Slowloris:

```
client_header_timeout 5s;
client_body_timeout 5s;
send_timeout 10s;
keepalive_timeout 15s;
```

- `client_header_timeout` y `client_body_timeout`: cierran conexiones que no envían datos en 5 segundos.
- `send_timeout`: evita que el servidor quede esperando indefinidamente al enviar respuestas.
- `keepalive_timeout`: define cuánto tiempo mantener viva una conexión HTTP inactiva.

4. Optimización del rendimiento con epoll y workers:

- En el bloque `events{}`, se especificó el uso del motor de eventos eficiente de Linux:

```
events {
    use epoll;
    worker_connections 4096;
}
```

- `epoll`: mecanismo moderno del kernel Linux para manejar miles de conexiones de forma no bloqueante.
- `worker_connections 4096`: permite que cada proceso worker maneje hasta 4096 conexiones simultáneas.

Ejecución de pruebas luego de implementar medidas

Prueba	Requests	Concurrency	Requests/s	Time/request (ms)	Errores
1	100	5	83	59.7	84
2	1000	50	234.92	212.8	969
3	5000	200	238.37	839	4886
4	10000	500	222	2249	9775
5	20000	1000	227	4404	19614

Cuadro 2: Resultados de pruebas de carga con ab

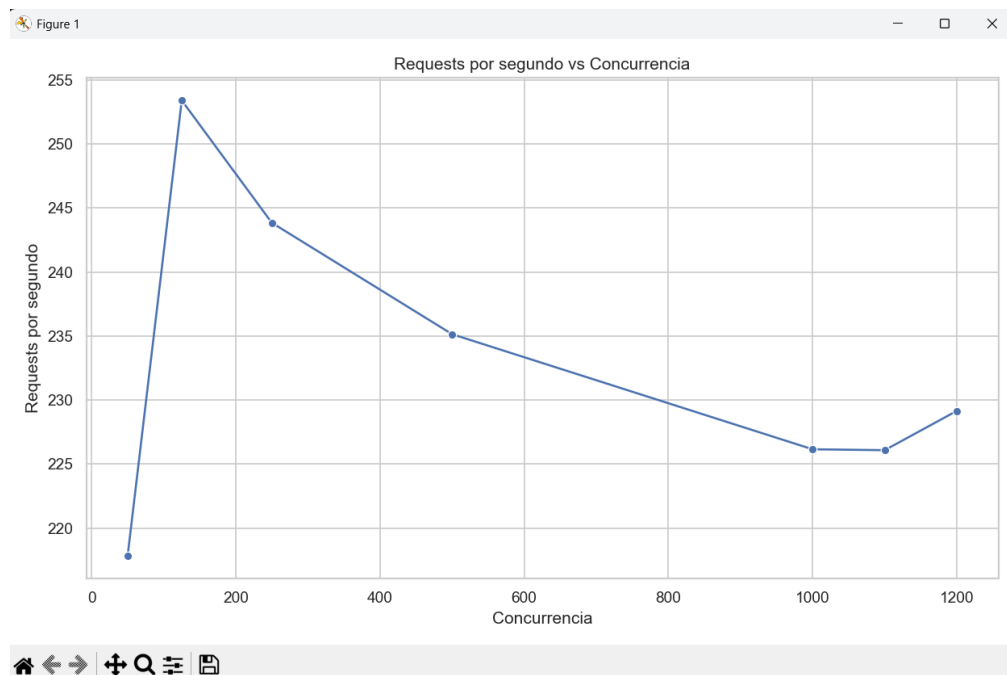


Figura 12: Requests por sec v/s concurrencia

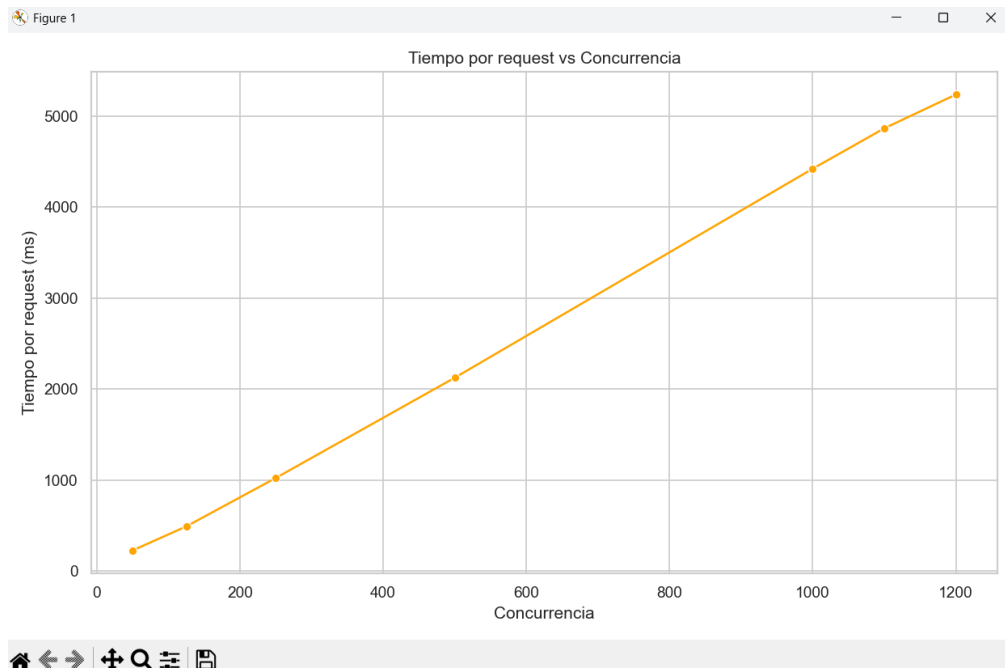


Figura 13: Tiempo por request v/s concurrencia

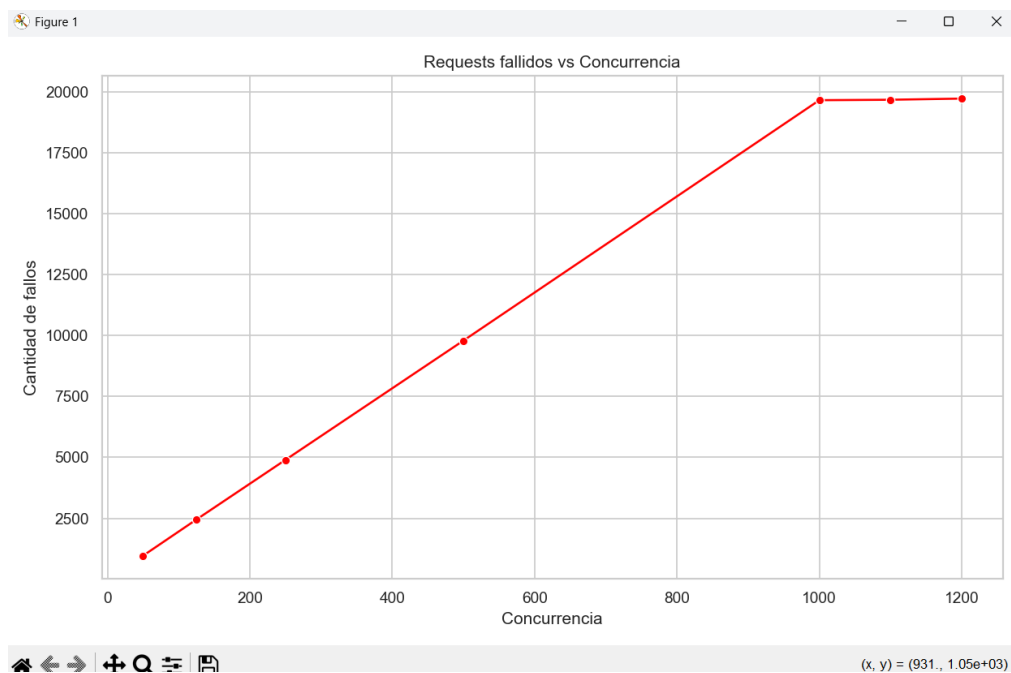


Figura 14: Requests fallidos v/s concurrencia

Logs


```

2025/07/03 07:08:21 [error] 204#204: *38288
limiting connections by zone "addr", client:
181.163.75.255, server: proyectoadmin.moood.com,
request: "GET /api/users/ HTTP/1.0", host:
"proyectoadmin.moood.com" 2025/07/03 07:08:21

```

Figura 15: Log de límite de conexiones desde una IP

```

2025/07/03 07:16:05 [error] 204#204: *75500
limiting requests, excess: 10.290 by zone
"mylimit", client: 181.163.75.255, server:
proyectoadmin.moood.com, request: "GET /api/users/
HTTP/1.0", host: "proyectoadmin.moood.com"

```

Figura 16: Log de límite de requests desde una IP

Carga del sistema

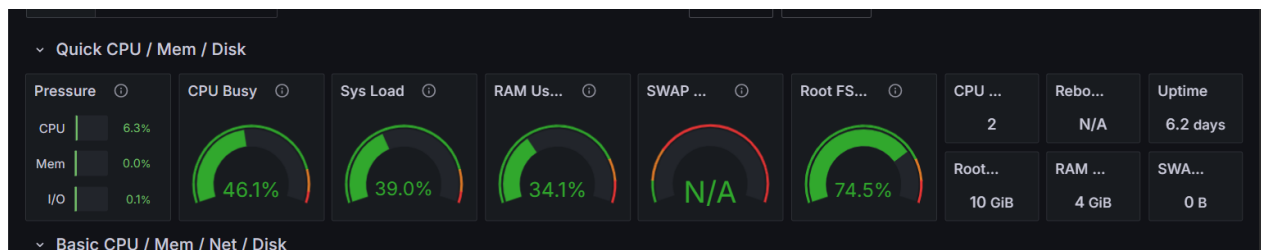


Figura 17: Carga del sistema con 20000 requests y concurrencia a 1000



Figura 18: Carga del sistema con 20000 requests y concurrencia a 1000

Hubo bajas significativas en la carga del sistema con test similares a los realizados antes de las modificaciones. El contenedor del proxy sigue siendo el más cargado, pero gracias al rechazo de múltiples conexiones y solicitudes de una IP, los contenedores de los servicios se ven protegidos, con una carga disminuida hacia ellos.

Análisis teórico: Detección de ataque DoS en entorno real

Un administrador de sistemas (SysAdmin) puede identificar que su infraestructura está siendo víctima de un ataque de denegación de servicio (DoS) mediante el monitoreo de ciertos indicadores clave de comportamiento anómalo en la red y los servicios. A continuación, se detallan las técnicas más utilizadas:

■ Análisis del tráfico de red:

- Uso de herramientas como `iftop`, `nload` o `Wireshark` para identificar picos inusuales de tráfico entrante hacia puertos específicos.
- Tráfico elevado proveniente de una o varias IPs puede ser una señal de ataque por saturación.

■ Monitoreo de recursos del sistema:

- Evaluar consumo anómalo de CPU, RAM y conexiones activas mediante comandos como `top`, `htop`, `vmstat`, o plataformas como Netdata y Grafana.
- Un aumento abrupto y sostenido en el uso de recursos, sin una carga esperada, puede indicar un ataque en curso.

■ Revisión de logs del servidor web:

- Examinar los archivos de log de Nginx (`access.log`, `error.log`) en busca de un número elevado de respuestas 429 (Too Many Requests), 502 o 504.
- Identificar patrones repetitivos de solicitudes desde IPs sospechosas.

■ Detección de conexiones persistentes:

- Utilizar `netstat`, `ss` o `conntrack` para listar y contar el número de conexiones activas.
- La presencia de cientos o miles de conexiones en estado `ESTABLISHED` o `TIME_WAIT` sin tráfico útil puede indicar ataques tipo Slowloris.

■ Alertas automáticas:

- Configurar herramientas como Fail2Ban, Suricata o sistemas SIEM para detectar comportamientos sospechosos y generar alertas automáticas.
- Estas herramientas pueden bloquear temporalmente IPs que generen tráfico malicioso repetitivo.

■ Comparación con patrones históricos:

- Utilizar dashboards históricos para comparar la actividad actual con el comportamiento normal del sistema.
 - Cambios abruptos en el número de peticiones por segundo (RPS) pueden indicar un ataque de saturación.
- **Revisión del estado de los servicios y procesos:** La revisión del estado de los servicios permite detectar sobrecargas. En el caso de servidores web como Nginx, mensajes como `‘‘worker_connections are not enough’’` o `‘‘reusing connections’’` en los logs pueden indicar que el servidor está manejando más conexiones de las que puede atender, lo cual es un síntoma típico de un ataque DoS.
 - **Disponibilidad de servicios:**
 - Monitoreo con Nagios, Zabbix, Prometheus.
 - Detección de lentitud inusual en servicios.

En resumen, un SysAdmin puede detectar un ataque DoS mediante una combinación de análisis de tráfico, monitoreo de recursos, revisión de logs y herramientas de alerta proactiva. La detección temprana permite actuar rápidamente para mitigar el impacto sobre la disponibilidad de los servicios.

3.3. Alta Disponibilidad con Réplicas

Para mejorar la resiliencia de la aplicación ante fallos y cargas elevadas, se implementó alta disponibilidad mediante el uso de Docker Swarm, desplegando múltiples réplicas de los microservicios principales. A continuación, se detallan los pasos seguidos:

1. Inicialización de Docker Swarm:

- Se ejecutó el comando `docker swarm init` para convertir la máquina en un nodo gestor del clúster Swarm.

2. Modificación del archivo `docker-compose.yml`:

- Se eliminó la directiva `container_name` de los servicios para permitir múltiples instancias.
- Se reemplazó `build:` por `image:` en los microservicios, ya que Swarm no soporta la opción `build`.
- Se agregó el bloque `deploy: replicas: 2` en los servicios `user-service` y `task-service` para especificar el número de réplicas.

3. Construcción de las imágenes de los servicios:

- Se construyeron las imágenes localmente con los comandos:


```
docker build -t user-service:latest ./user-service
docker build -t task-service:latest ./task-service
```

4. Despliegue del stack con Docker Swarm:

- Se desplegó el stack ejecutando:

```
docker stack deploy -c docker-compose.yml proyecto
```

5. Verificación de réplicas activas:

- Se ejecutó `docker service ls` para comprobar que cada servicio tuviera dos réplicas activas.

6. Simulación de fallos y pruebas de resiliencia:

- Se eliminó manualmente una réplica con `docker container rm -f <ID>` y se observó cómo Swarm la reponía automáticamente.
- Se verificó que el servicio seguía respondiendo correctamente sin interrupciones.

7. Pruebas de carga y comparación con DoS:

- Se repitieron los ataques DoS realizados anteriormente.
- Se comprobó mediante logs y comportamiento de la aplicación que la presencia de múltiples réplicas mejoraba la tolerancia al fallo y la capacidad de respuesta ante múltiples peticiones simultáneas.

Evidencia

Prueba	Requests	Concurrency	Requests/s	Time/request (ms)	Errores
1	100	5	78	63.7	83
2	1000	50	205	243.8	966
3	5000	200	245	814	4889
4	10000	500	229	2181	9776
5	20000	1000	222	4499	19616

Cuadro 3: Resultados de pruebas de carga con `ab`

```
diegosalvador01032003@mv-admin-proy2-1:~/ProyectoAdmin$ sudo docker ps
```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
f86c7703eb2e	jc21/nginx-proxy-manager:latest	"/init"		22 minutes ago	Up 22 minutes (healthy)	80-81/tcp, 443
6d769c4a32f7	task-service:latest	projecto_npm.1.sagddrl077fz7m6s6xhsmjfk	"docker-entrypoint.s..."	22 minutes ago	Up 22 minutes (healthy)	3000/tcp
f177e108f009	task-service:latest	projecto_task-service.1.19skyp703cahnt5pe0q771pke	"docker-entrypoint.s..."	22 minutes ago	Up 22 minutes (healthy)	3000/tcp
b9f5ea573f5b	user-service:latest	projecto_task-service.2.3zql1rnmpxm42kop3q5hbrmz0	"docker-entrypoint.s..."	22 minutes ago	Up 22 minutes (healthy)	3000/tcp
6a65205386aa	user-service:latest	projecto_user-service.2.3mhbbis3wbsk9uvcqbms4w5da	"docker-entrypoint.s..."	22 minutes ago	Up 22 minutes (healthy)	3000/tcp
316eddc0f13a	postgres:15	projecto_db-tasks.1.zoptukcileislenkxh9tz3381	"docker-entrypoint.s..."	24 minutes ago	Up 24 minutes (healthy)	5432/tcp
13856644c759	postgres:15	projecto_db-users.1.irhfw0so2h7umrew4yokw9xs	"docker-entrypoint.s..."	24 minutes ago	Up 24 minutes (healthy)	5432/tcp
a4287a02f3f5	docker.elastic.co/beats/filebeat:8.8.0	filebeat	"/usr/bin/tini -- /u..."	39 hours ago	Up 39 hours	
2790d98507ce	gcr.io/cadvisor/cadvisor:v0.47.2	cadvisor	"/usr/bin/cadvisor -..."	38 hours ago	Up 17 minutes (healthy)	
712459daf64d	prom/node-exporter:latest	node-exporter	"/bin/node_exporter ..."	38 hours ago	Up 38 hours	0.0.0.0:9100->

```
diegosalvador01032003@mv-admin-proy2-1:~/ProyectoAdmin$
```

Figura 19: docker ps - réplicas funcionando

```
ssh.cloud.google.com/v2/ssh/projects/gentle-charmer-463402-n6/zones/southamerica-west1-b/instances/mv-admin-proy2-17a...
```

```
ssh.cloud.google.com/v2/ssh/projects/gentle-charmer-463402-n6/zones/southamerica-west1-b/instances/mv-admin...
```

SSH en el navegador

```
2790d98507ce gcr.io/cadvisor/cadvisor:v0.47.2 "/usr/bin/cadvisor -..." 38 hours ago
inutes (healthy)
712459daf64d prom/node-exporter:latest "/bin/node_exporter ..." 38 hours ago
ours 0.0.0.0:9100->9100/tcp, :::9100->9100/tcp node-exporter
diegosalvador01032003@mv-admin-proy2-1:~/ProyectoAdmin$
diegosalvador01032003@mv-admin-proy2-1:~/ProyectoAdmin$ docker service scale projecto_user-service
projecto_user-service scaled to 1
overall progress: 1 out of 1 tasks
1/1: running
verify: Service converged
diegosalvador01032003@mv-admin-proy2-1:~/ProyectoAdmin$ sudo docker ps
```

CONTAINER ID	IMAGE	PORTS	COMMAND	NAMES	CREATED
8d7187de8468	user-service:latest		"docker-entrypoint.s..."	projecto_user-service.1.2541oga	32 minutes ago
inutes (healthy)		3000/tcp			
u3j0k5n76w	postgres:15		"docker-entrypoint.s..."	projecto_db-tasks.1.qd5uupadiqv	37 minutes ago
fed1b8474c4f	postgres:15	5432/tcp			
inutes (healthy)					
zcpxp25	postgres:15		"docker-entrypoint.s..."	projecto_db-users.1.rh5n2ac9ji7	37 minutes ago
inutes (healthy)		5432/tcp			
39m6mt	task-service:latest		"docker-entrypoint.s..."	projecto_task-service.1.sap06x1	37 minutes ago
329c689008e3	task-service:latest	3000/tcp			
inutes (healthy)					
yrlv0czvu0	task-service:latest		"docker-entrypoint.s..."	projecto_task-service.2.b9p8t3c	37 minutes ago
799789e76e76	task-service:latest	3000/tcp			
inutes (healthy)					
ns3fa8e1d8	jc21/nginx-proxy-manager:latest		"/init"	projecto_npm.1.ulqbynkqriqvendg	37 minutes ago
489a3938622c	jc21/nginx-proxy-manager:latest	80-81/tcp, 443/tcp			
inutes (healthy)					
m	docker.elastic.co/beats/filebeat:8.8.0		"/usr/bin/tini -- /u..."		39 hours ago
a4287a02f3f5	docker.elastic.co/beats/filebeat:8.8.0		filebeat		
ours	gcr.io/cadvisor/cadvisor:v0.47.2		"/usr/bin/cadvisor -..."		39 hours ago
2790d98507ce	gcr.io/cadvisor/cadvisor:v0.47.2		cadvisor		
t an hour (healthy)	prom/node-exporter:latest		"/bin/node_exporter ..."		39 hours ago
712459daf64d	prom/node-exporter:latest	0.0.0.0:9100->9100/tcp, :::9100->9100/tcp	node-exporter		
ours					

```
diegosalvador01032003@mv-admin-proy2-1:~/ProyectoAdmin$
```

Workspaces

GET Listar usuarios

Pruebas / user-service / Listar usuarios

GET {{base_url}} /api/users

Params Auth Headers (6) Body Scripts Settings

Query Params

Key	Value
Key	Value

Body 200 OK 49 ms

```
{
  "status": 200,
  "message": "OK",
  "data": []
}
```

Figura 20: Se elimina una réplica de user-service y sigue funcionando

```
diegosalvador01032003@mv-admin-proy2-1:~/ProyectoAdmin$ sudo docker ps
```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
8d7187de8468	user-service:latest	proyecto_user-service.1.2541oga5t7h1lr0u3j0k5n76w	"docker-entrypoint.s..."	25 seconds ago	Up 21 seconds (healthy)	3000/tcp
fed1b8474c4f	postgres:15	proyecto_db-tasks.1.qd5uupadigvo96fxi6lzcxpz5	"docker-entrypoint.s..."	5 minutes ago	Up 5 minutes (healthy)	5432/tcp
1c6eeb26f379	postgres:15	proyecto_db-users.1.rh5nzac9ji7ybqwdlk739m6mt	"docker-entrypoint.s..."	5 minutes ago	Up 5 minutes (healthy)	5432/tcp
329c689008c3	task-service:latest	proyecto_task-service.1.sap06xl6tc0k4rvyrlv0czvu0	"docker-entrypoint.s..."	5 minutes ago	Up 5 minutes (healthy)	3000/tcp
799789e76e76	task-service:latest	proyecto_task-service.2.b9p8t3crj0hgdtzns3fa8e1d8	"docker-entrypoint.s..."	5 minutes ago	Up 5 minutes (healthy)	3000/tcp
cfce3cf5f901	user-service:latest	proyecto_user-service.2.6a7x6nr3cultcn0xvoo215j15	"docker-entrypoint.s..."	5 minutes ago	Up 5 minutes (healthy)	3000/tcp
489a3938622c	jc21/nginx-proxy-manager:latest	proyecto_npm.1.ulqbynkqrivjendgj6uli98fm	"/init"	5 minutes ago	Up 5 minutes (healthy)	80-81/tcp, 443
/tcp	proyecto_npm.1.ulqbynkqrivjendgj6uli98fm					
a4287a02f3f5	docker.elastic.co/beats/filebeat:8.8.0	filebeat	"/usr/bin/tini -- /u..."	38 hours ago	Up 38 hours	
2790d98507ce	gcr.io/cadvisor/cadvisor:v0.47.2	cadvisor	"/usr/bin/cadvisor -..."	38 hours ago	Up 39 minutes (healthy)	
712459daf64d	prom/node-exporter:latest	node-exporter	"/bin/node_exporter ..."	38 hours ago	Up 38 hours	0.0.0.0:9100->
9100/tcp, :::9100->9100/tcp	node-exporter					

```
diegosalvador01032003@mv-admin-proy2-1:~/ProyectoAdmin$
```

Figura 21: Se elimina un contenedor - reinicio automático

El tiempo de recuperación fue casi instantáneo, con una duración aprox. de 5 a 10 seg.

```
diegosalvador01032003@mv-admin-proy2-1:~/ProyectoAdmin$ sudo docker logs cfce3cf5f901
```

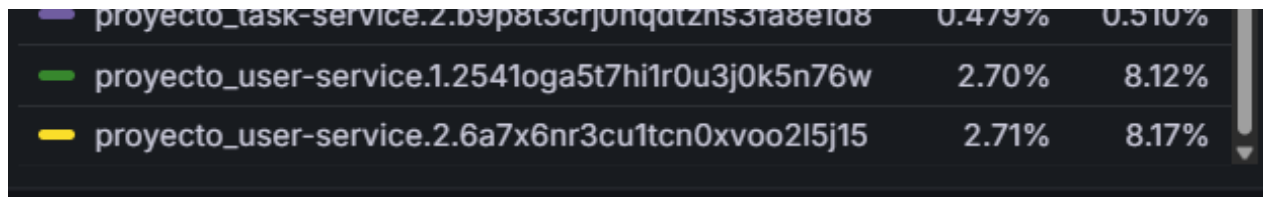
```
> user-service@1.0.0 start
> node index.js
```

```
Intentando conectar a la base de datos... (intento 1)
Error de conexión: getaddrinfo ENOTFOUND db-users
Intentando conectar a la base de datos... (intento 2)
Error de conexión: getaddrinfo ENOTFOUND db-users
Intentando conectar a la base de datos... (intento 3)
Error de conexión: getaddrinfo ENOTFOUND db-users
Intentando conectar a la base de datos... (intento 4)
Error de conexión: getaddrinfo ENOTFOUND db-users
Intentando conectar a la base de datos... (intento 5)
Base de datos conectada
user-service corriendo en puerto 3000
[2025-07-03T16:11:05.882Z] /users llamado desde contenedor: cfce3cf5f901
[2025-07-03T16:13:49.078Z] /users llamado desde contenedor: cfce3cf5f901
[2025-07-03T16:13:50.605Z] /users llamado desde contenedor: cfce3cf5f901
[2025-07-03T16:13:51.789Z] /users llamado desde contenedor: cfce3cf5f901
[2025-07-03T16:14:48.635Z] /users llamado desde contenedor: cfce3cf5f901
diegosalvador01032003@mv-admin-proy2-1:~/ProyectoAdmin$ sudo docker logs 7755861e116a
```

```
> user-service@1.0.0 start
> node index.js
```

```
Intentando conectar a la base de datos... (intento 1)
Error de conexión: getaddrinfo ENOTFOUND db-users
Intentando conectar a la base de datos... (intento 2)
Error de conexión: getaddrinfo ENOTFOUND db-users
Intentando conectar a la base de datos... (intento 3)
Error de conexión: getaddrinfo ENOTFOUND db-users
Intentando conectar a la base de datos... (intento 4)
Error de conexión: getaddrinfo ENOTFOUND db-users
Intentando conectar a la base de datos... (intento 5)
Base de datos conectada
user-service corriendo en puerto 3000
[2025-07-03T16:13:48.195Z] /users llamado desde contenedor: 7755861e116a
[2025-07-03T16:13:49.947Z] /users llamado desde contenedor: 7755861e116a
```

Figura 22: Logs que muestran balanceo de carga efectivo



proyecto_task-service.2.b9p8t3c1j0nqdtzns31a8e1d8	0.479%	0.510%
proyecto_user-service.1.2541oga5t7hi1r0u3j0k5n76w	2.70%	8.12%
proyecto_user-service.2.6a7x6nr3cu1tcn0xvoo2l5j15	2.71%	8.17%

Figura 23: La carga se distribuye entre contenedores, pasando de un contenedor a 20 % a dos en 8 %

Conclusión: Con esta configuración, la aplicación quedó preparada para tolerar fallos individuales de instancias, mejorando la disponibilidad y robustez del sistema frente a escenarios de carga y ataque.