

MANUAL 9: GUÍA DE TROUBLESHOOTING AVANZADO - SISTEMA CONA

INFORMACIÓN DEL DOCUMENTO

Fecha de Creación: 21 de Julio de 2025

Proyecto: Sistema CONA (Gestión CONAVEG)

Audiencia: DevOps, Desarrolladores Senior, SRE

Nivel: Experto

Tiempo Estimado: 3-5 horas (estudio y práctica)

Última Actualización: 21 de Julio de 2025

OBJETIVOS DE APRENDIZAJE

Al finalizar este manual, serás capaz de:

- ☒ Realizar diagnósticos avanzados de problemas complejos de performance y seguridad.
 - ☒ Utilizar herramientas de debugging y profiling para analizar la JVM.
 - ☒ Interpretar logs y eventos de auditoría para reconstruir incidentes.
 - ☒ Resolver problemas de optimización de base de datos y performance de la aplicación.
 - ☒ Liderar la respuesta técnica a incidentes de seguridad.
 - ☒ Ejecutar procedimientos de recuperación de desastres.
 - ☒ Establecer y seguir procedimientos de escalación técnica.
-

INTRODUCCIÓN AL TROUBLESHOOTING AVANZADO

Esta guía va más allá de los problemas comunes cubiertos en otros manuales. Está diseñada para situaciones críticas donde las soluciones estándar no son suficientes. Aquí nos sumergiremos en el análisis profundo de la JVM, la base de datos y los patrones de seguridad para resolver los problemas más desafiantes.

Requisito fundamental: Antes de usar esta guía, asegúrate de haber consultado la [docs/testing/Troubleshooting_Guide.md](#) para descartar problemas comunes.

☐ HERRAMIENTAS DE DIAGNÓSTICO AVANZADO

Para un análisis profundo, necesitarás herramientas que te permitan inspeccionar el estado interno de la aplicación y el servidor.

Herramientas de Línea de Comandos (JDK):

- `jps`: Lista los procesos de Java en ejecución. Útil para obtener el `PID` de la aplicación.

- **jstat**: Monitorea las estadísticas de la JVM, especialmente la recolección de basura (Garbage Collection).
- **jmap**: Genera un *heap dump* (una instantánea de la memoria) para análisis de memory leaks.
- **jstack**: Genera un *thread dump* para analizar el estado de todos los hilos, útil para detectar deadlocks.

Herramientas de Profiling (Visuales):

- VisualVM: Una herramienta visual todo-en-uno que combina las funcionalidades de las herramientas de línea de comandos y permite un profiling en tiempo real.
- YourKit / JProfiler: Profilers comerciales muy potentes para un análisis de performance aún más detallado.



ESCENARIO 1: PROBLEMAS DE PERFORMANCE Y MEMORIA

Estos son los problemas más críticos y difíciles de diagnosticar.

Caso 1.1: Alto Uso de CPU Inexplicado

Síntomas: La CPU del servidor está constantemente por encima del 90%, pero el tráfico de la aplicación no es inusualmente alto.

Diagnóstico:

1. Identificar el Proceso: Usa **top** y luego **jps** para confirmar que el proceso Java de CONA es el culpable.
2. Identificar el Hilo Culpable:

```
# Obtén el PID del proceso Java
PID=$(pgrep java)
# Muestra los hilos ordenados por uso de CPU
top -H -p $PID
```

Anota el **PID** del hilo que consume más CPU.

3. Convertir el PID del Hilo a Hexadecimal:

```
printf '%x\n' <PID_DEL_HILO>
```

4. Generar un Thread Dump:

```
jstack $PID > /tmp/threaddump.txt
```

5. Buscar el Hilo en el Dump: Busca el PID en formato hexadecimal (obtenido en el paso 3) dentro de **threaddump.txt**. El stack trace asociado a ese hilo te dirá exactamente qué método está atascado en un bucle o consumiendo la CPU.

Solución Común: A menudo, esto es causado por una query ineficiente, un bucle infinito en el código o un problema con una librería externa. El stack trace te dará la pista definitiva para que el equipo de desarrollo lo solucione.

Caso 1.2: Memory Leak (Fuga de Memoria)

Síntomas: El uso de memoria de la aplicación crece constantemente con el tiempo, incluso con carga baja, y eventualmente causa un `OutOfMemoryError`.

Diagnóstico:

1. Monitorear la Recolección de Basura (GC):

```
# Monitorea el GC cada 5 segundos, 10 veces
jstat -gc $PID 5s 10
```

Si ves que el uso del "Old Gen" (O) nunca disminuye significativamente después de un Full GC (FGC), es un fuerte indicio de un memory leak.

2. Generar un Heap Dump:

```
# Genera un volcado de la memoria. ¡CUIDADO! Esto puede pausar la aplicación brevemente.
jmap -dump:format=b,file=/tmp/heapdump.hprof $PID
```

3. Analizar el Heap Dump:

- Transfiere el archivo `heapdump.hprof` a tu máquina local.
- Usa una herramienta como Eclipse Memory Analyzer (MAT) o VisualVM para abrir el dump.
- Estas herramientas te mostrarán qué objetos están ocupando la mayor parte de la memoria y no pueden ser recolectados por el GC, apuntando directamente a la causa del leak.

Solución Común: Los memory leaks suelen ser causados por objetos que se añaden a colecciones estáticas y nunca se eliminan, o por recursos (como conexiones o streams) que no se cierran correctamente.

ESCENARIO 2: INCIDENTES DE SEGURIDAD COMPLEJOS

Caso 2.1: Detección de un Posible Acceso No Autorizado

Síntomas: Se observa actividad sospechosa en los logs, como un administrador iniciando sesión desde una IP desconocida fuera del horario laboral.

Diagnóstico Forense:

4. Análisis de Logs de Auditoría:

- Objetivo: Reconstruir la línea de tiempo de las acciones del atacante.
- Acción: Extraer todos los eventos de `security_audit_logs` para el usuario y la IP sospechosos.

```
SELECT * FROM security_audit_logs
WHERE email = 'usuario.sospechoso@conaveg.com' OR ip_address = '1.2.3.4'
ORDER BY timestamp ASC;
```

Busca patrones: ¿qué endpoints accedió? ¿Hubo intentos fallidos antes del éxito? ¿Qué datos consultó o modificó?

5. Correlación con Logs de Aplicación:

- Objetivo: Obtener más contexto técnico sobre las solicitudes.
- Acción: Usa el `timestamp` de los eventos de auditoría para buscar en `logs/spring.log` y `logs/access.log` (si está configurado). Esto puede revelar los `User-Agent`, parámetros exactos de la solicitud y posibles errores que el atacante provocó.

6. Análisis de la Base de Datos:

- Objetivo: Evaluar el impacto. ¿Qué datos fueron modificados?
- Acción: Si tienes auditoría a nivel de tabla (ej. con triggers), revisa los cambios. Si no, tendrás que revisar manualmente las tablas relevantes (`proyectos`, `inventario`, etc.) en busca de modificaciones en la ventana de tiempo del incidente.

Respuesta al Incidente:

7. Contención:

- Invalidar la sesión del usuario: Forzar el cierre de sesión.
- Bloquear la cuenta del usuario: Cambiar su estado a `BLOQUEADO`.
- Bloquear la IP a nivel de firewall: `sudo ufw insert 1 deny from 1.2.3.4`.

8. Erradicación:

- Forzar el cambio de contraseña de la cuenta comprometida.
- Revisar si el atacante creó otras cuentas o modificó permisos.

9. Recuperación:

- Restaurar los datos modificados desde un backup si es necesario.
- Comunicar el incidente según las políticas de la empresa.

ESCENARIO 3: RECUPERACIÓN DE DESASTRES

Caso 3.1: Corrupción Total de la Base de Datos

Síntomas: La base de datos no inicia, los archivos de datos están corruptos y no se puede acceder a la aplicación.

Procedimiento de Recuperación: Este procedimiento asume que tienes backups diarios automáticos como se describe en el "Manual de Despliegue y DevOps".

10. Declarar el Desastre y Comunicar: Informar a los stakeholders que el sistema está caído y que se está iniciando el proceso de recuperación.

11. Detener la Aplicación: Asegurarse de que el contenedor `cona-app` esté detenido para evitar que intente conectarse a una base de datos corrupta.

```
docker-compose stop cona-app
```

12. Destruir la Base de Datos Corrupta:

```
# Detener y eliminar el contenedor de la BD
docker-compose stop cona-db
docker-compose rm -f cona-db
# ¡CUIDADO! Esto elimina permanentemente el volumen de datos corrupto.
docker volume rm cona_db_data
```

13. Restaurar desde el Último Backup Válido:

- Identifica el archivo de backup más reciente y confiable en tu directorio de backups.
- Levanta un nuevo contenedor de base de datos vacío:

```
docker-compose up -d cona-db
# Espera unos segundos a que se inicialice
sleep 20
```

- Ejecuta el script de restauración:

```
gunzip < /opt/cona/backups/LATEST_BACKUP.sql.gz | docker-compose exec -T cona-db mysql -u root -p'your_strong_root_password' conaveg_db
```

14. Verificar la Integridad de los Datos Restaurados:

- Conéctate a la base de datos y realiza algunas consultas de verificación.

```
SELECT COUNT(*) FROM empleados;
SELECT * FROM proyectos ORDER BY id DESC LIMIT 5;
```

15. Iniciar la Aplicación y Validar:

```
docker-compose up -d cona-app
```

- Realiza una validación funcional completa del sistema.


16. Análisis Post-Mortem: Investigar la causa raíz de la corrupción para prevenir futuras ocurrencias.


PROCEDIMIENTOS DE ESCALACIÓN


Cuando un problema supera tu capacidad de resolución, es crucial tener un plan de escalación claro.


- Nivel 1 (Tú: DevOps/Desarrollador Senior):
 - Tiempo Límite: 1 hora para diagnosticar y resolver.
 - Acción: Si no hay progreso o la causa raíz no está clara después de 1 hora, escala.
- Nivel 2 (Equipo de Arquitectura / Líder Técnico):
 - Cuándo escalar:

- Si el problema afecta a múltiples sistemas.
- Si se sospecha de un fallo de diseño fundamental.
- Si se requiere una decisión que impacte la arquitectura (ej. cambiar de tipo de base de datos).
- Información a proveer: Resumen del incidente, logs relevantes, dumps (si los hay), y las hipótesis que has descartado.
- Nivel 3 (Soporte de Proveedores Externos):
 - Cuándo escalar:
 - Si el problema parece estar en un servicio en la nube (AWS, Azure, etc.).
 - Si una librería de terceros está causando un error crítico.
 - Acción: Abrir un ticket de soporte con toda la evidencia técnica recopilada.

 Fecha de Creación: 21 de Julio de 2025

 Responsable: Equipo de Arquitectura y SRE CONA

 Estado: Manual Completo y Validado

 Próxima Revisión: 21 de Agosto de 2025