Instalación KOHA ILS 23.05 y MariaDB

Informe Técnico

Sumario

Documentación de Instalación: Servidor Koha 23.05 sobre Debian 12

Requisitos Previos

- Servidor con Debian 12 (Bookworm) con instalación limpia.
- Acceso a la terminal con privilegios de administrador (root o mediante sudo).
- Conexión a Internet activa para la descarga de paquetes.

Paso 1: Preparación del Sistema Operativo

Antes de comenzar con la instalación específica, es fundamental asegurar que el sistema base esté completamente actualizado e instalar las herramientas necesarias para la gestión de repositorios. Ejecuta las siguientes instrucciones en la terminal:

- sudo apt update
- sudo apt upgrade -y
- sudo apt install -y wget gnupg software-properties-common default-mysql-server

Paso 2: Configuración del Repositorio de Koha

Debian 12 ha deprecado el uso tradicional de apt-key. Por lo tanto, utilizaremos el método moderno con directorios "keyrings" para añadir la llave GPG oficial del proyecto Koha y configurar el repositorio para la versión 23.05.

1. Crea el directorio para las llaves si no existe:

- sudo mkdir -p /etc/apt/keyrings

2. Descarga e instala la llave GPG de Koha:

- wget -qO - https://debian.koha-community.org/koha/gpg.asc | sudo tee /etc/apt/keyrings/koha.asc > /dev/null

3. Añade el repositorio de la rama 23.05 a tus orígenes de software:

- echo "deb [signed-by=/etc/apt/keyrings/koha.asc] http://debian.koha-community.org/koha 23.05 main" | sudo tee /etc/apt/sources.list.d/koha.list

4. Actualiza el índice de paquetes para reconocer el nuevo repositorio:

- sudo apt update

Paso 3: Instalación del Paquete Base

Procede con la instalación del paquete principal de Koha. Este proceso resolverá e instalará automáticamente todas las dependencias necesarias de Perl y las utilidades del sistema requeridas.

- sudo apt install -y koha-common

Paso 4: Configuración de Puertos y Apache

Koha requiere que el servidor web Apache escuche en puertos específicos para mantener separados el catálogo público (OPAC) y la interfaz de administración (Intranet). En este ejemplo, el OPAC utilizará el puerto 80 y la Intranet el puerto 8080.

1. Abre el archivo de configuración de sitios de Koha con tu editor de texto preferido (por ejemplo, nano):

- sudo nano /etc/koha/koha-sites.conf

2. Asegúrate de que las directivas de puertos estén definidas de la siguiente manera:

- INTRAPORT="8080"
- OPACPORT="80"

3. Habilita los puertos en Apache editando el archivo ports.conf:

- sudo nano /etc/apache2/ports.conf
- (Añade o verifica que existan las líneas: Listen 80 y Listen 8080)

4. Habilita los módulos requeridos por Koha en Apache y reinicia el servicio:

- sudo a2enmod rewrite cgi headers proxy_http
- sudo systemctl restart apache2

Paso 5: Creación de la Instancia de Koha

Con el entorno base preparado, el siguiente paso es crear la instancia del catálogo. Para este documento, utilizaremos el nombre de instancia "fadu". Este comando creará la base de datos de forma automática, generará los archivos de configuración pertinentes y configurará el VirtualHost en Apache.

- sudo koha-create --create-db fadu

Paso 6: Activación del Sitio web

Una vez creada la instancia, debes habilitar el sitio específico en Apache y recargar el servicio para aplicar todos los cambios de configuración.

- sudo a2ensite fadu
- sudo systemctl reload apache2

Paso 7: Obtención de Credenciales para el Instalador Web

Para finalizar la instalación a través de la interfaz gráfica web, necesitarás el usuario y la contraseña maestra que el sistema ha generado automáticamente durante la creación de la instancia.

1. El nombre de usuario por defecto siempre sigue la estructura koha_[nombre_instancia]. En este caso será: koha_fadu

2. Para obtener la contraseña autogenerada, ejecuta:

- sudo koha-passwd fadu (Nota importante: Resguarda estas credenciales, ya que serán indispensables para el primer ingreso al sistema).

Paso 8: Configuración Final vía Web

1. Abre un navegador web en cualquier equipo de la red e ingresa a la dirección IP o dominio del servidor especificando el puerto de Intranet (por ejemplo: http://IP_DEL_SERVIDOR:8080).
2. Inicia sesión con el usuario koha_fadu y la contraseña obtenida en el paso anterior.
3. Sigue las instrucciones del asistente web para instalar las tablas estructurales de la base de datos, seleccionar el formato MARC (MARC21 o UNIMARC) e importar los datos de configuración básicos requeridos.

Bitácora de Servidor: Migración de Motor de Búsqueda y Actualización de Catálogo

Sistema: Koha 23.05 sobre Debian

Objetivo: Migración del motor de búsqueda de Zebra a Elasticsearch para mejorar el rendimiento y las capacidades de búsqueda, y ajustes de visualización en el OPAC.

---

1. Resumen de la Intervención

Se llevó a cabo la transición del motor de indexación y búsqueda del sistema integrado de gestión de bibliotecas (Koha) de Zebra a Elasticsearch. El proceso se diseñó para mantener el sistema operativo durante la configuración inicial, reduciendo el tiempo de inactividad a cero durante la reindexación masiva. Adicionalmente, se implementó una regla de visualización condicional (XSLT) para el OPAC.

2. Instalación y Configuración del Motor

- Selección de Versión: Se determinó que la versión más estable y compatible con Koha 23.05 es Elasticsearch 7.17. Se procedió a limpiar intentos previos con la versión 8.x para evitar conflictos de compatibilidad estructural en los índices.
- Configuración del Servicio (`elasticsearch.yml`):
    - Se configuró el entorno local (`network.host: 127.0.0.1`, `http.port: 9200`).
    - Se estableció la topología de nodo único (`discovery.type: single-node`).
    - Se resolvieron conflictos de configuración deshabilitando parámetros incompatibles de clúster (`cluster.initial_master_nodes`).

3. Resolución de Dependencias (Módulo Perl)

Se identificó un error crítico de compatibilidad entre la versión de la librería Perl instalada por los repositorios de Debian (`libsearch-elasticsearch-perl` versión 8.00) y el código base de Koha 23.05, lo que producía el error de ejecución: `Not a HASH reference at Elasticsearch.pm line 1338` al intentar verificar el estado del índice.

- Solución Aplicada:
    1. Se inhabilitó la librería v8.00 del sistema renombrando los directorios en `/usr/share/perl5/Search/Elasticsearch`.
    2. Se forzó la compilación e instalación de la versión compatible 6.81 utilizando `cpanm` (`cpanm --force Search::Elasticsearch@6.81`).

4. Configuración de Koha (`koha-conf.xml`)

Se detectó y corrigió un error de sintaxis en el archivo de configuración de la instancia de Koha.

- Problema: El bloque de configuración `<elasticsearch>` se encontraba fuera de la etiqueta raíz `<config>`, lo que impedía que el parser XML de Koha cargara los parámetros del servidor, derivando en un fallo al instanciar el cliente.
- Solución: Se reubicó el bloque `<elasticsearch>` correctamente dentro de las etiquetas `<config>`, restableciendo la comunicación entre Koha y el servicio en el puerto 9200 para la instancia `bibliofadu`.

5. Instalación de Componentes de Análisis (ICU)

Durante el primer intento de creación del índice (`koha-rebuild-elasticsearch`), el sistema abortó devolviendo un error `[illegal_argument_exception]` por no encontrar el filtro `icu_folding`.

- Solución: Se instaló el plugin oficial de soporte Unicode para Elasticsearch, necesario para el manejo correcto de caracteres especiales y acentos en Koha.
    - Comando ejecutado: `sudo /usr/share/elasticsearch/bin/elasticsearch-plugin install analysis-icu`
    - Se reinició el servicio Elasticsearch para aplicar los cambios.

6. Proceso de Indexación y Activación

1. Indexación Inicial (Bulk): Se ejecutó el volcado masivo de registros de la base de datos hacia Elasticsearch en segundo plano utilizando el comando: `koha-shell bibliofadu -c "perl /usr/share/koha/bin/search_tools/rebuild_elasticsearch.pl -b -r -v"`
2. Activación: Una vez finalizada la indexación, se modificó la preferencia del sistema `SearchEngine` en la interfaz administrativa de Koha, pasando de Zebra a Elasticsearch.
3. Reinicio de Servicios: Se reiniciaron los servicios web para limpiar la caché y forzar la lectura del nuevo motor.

7. Modificación de Hojas de Estilo XSLT (OPAC)

Aprovechando la ventana de mantenimiento, se ajustó la visualización de la leyenda "Pedir por:" (basada en el campo MARC 001) para que sea exclusiva de los registros correspondientes a libros.

- Modificación: Se editó el XSLT inyectando una validación sobre el campo *Leader* (Cabecera), verificando específicamente la posición 06 (Material textual, valor 'a') y la posición 07 (Monografía, valor 'm').

8. Actualización de Tareas Programadas (Cronjobs)

Para garantizar la actualización continua del nuevo catálogo y evitar el solapamiento de procesos de indexación obsoletos, se reconfiguró el crontab del usuario del sistema (`koha_bibliofadu`).

- Desactivación de Zebra: Se comentaron (inhabilitaron) las líneas correspondientes a la ejecución periódica de `rebuild_zebra.pl`.
- Activación de Elasticsearch: Se añadió la directiva para la indexación incremental de Elasticsearch, programada para ejecutarse cada 5 minutos, optimizando la disponibilidad de los nuevos registros.
    - Comando programado: `*/5 * * * * /usr/share/koha/bin/search_tools/rebuild_elasticsearch.pl -r -v -p 2 -n -q >/dev/null 2>&1`

Bitácora de Servidor: Optimización y Configuración de Koha 23.05 con Elasticsearch

Sistema: Koha ILS (Versión 23.05) sobre Debian Linux

Instancia: bibliofadu

Motor de Búsqueda: Elasticsearch (Nodo único)

1. Resumen de Intervención

Se realizó una auditoría y reconfiguración integral del subsistema de búsqueda tras la migración del motor Zebra a Elasticsearch. El objetivo de la intervención fue optimizar el consumo de recursos, garantizar la indexación en tiempo real de los registros bibliográficos y establecer políticas de resguardo (backups) eficientes.

2. Reestructuración de Tareas Programadas (Cronjobs)

Se eliminó la dependencia de tareas cron para la indexación rutinaria, delegando esta función al demonio residente.

Acciones realizadas en el Crontab:

- Eliminación: Se removieron las entradas correspondientes a `rebuild_zebra.pl` para evitar el consumo innecesario de CPU y doble indexación.
- Mantenimiento conservado: Se mantuvieron los scripts críticos de Koha con las siguientes frecuencias:
    - Procesamiento de cola de correos (`process_message_queue.pl`): Cada 15 minutos.
    - Cálculo de multas y avisos de vencimiento (`fines.pl`, `overdue_notices.pl`): Ejecución nocturna diaria.
    - Limpieza de base de datos (`cleanup_database.pl --sessions --sessdays 7...`): Ejecución semanal para evitar el crecimiento desmedido de tablas temporales.
    - Generación de Sitemap (`sitemap.pl`): Ejecución nocturna.

3. Optimización del Clúster Elasticsearch

Al operar en una arquitectura de nodo único, Elasticsearch reportaba por defecto un estado de clúster `YELLOW` debido a la espera de un nodo secundario para asignar réplicas.

Acciones realizadas:

- Se reconfiguró el índice en caliente para requerir `0` réplicas, pasando el estado del clúster a `GREEN`.
    - *Comando ejecutado:* `curl -XPUT 'localhost:9200/*/_settings' -H 'Content-Type: application/json' -d '{"index": {"number_of_replicas": 0}}'`
- Protección del paquete: Se bloqueó la actualización automática de Elasticsearch vía APT para evitar rupturas de compatibilidad futuras con la versión actual de Koha.
    - *Comando ejecutado:* `apt-mark hold elasticsearch`

4. Auditoría de Indexación en Tiempo Real (Demonio)

Se verificó el estado del servicio `koha-common` mediante `systemctl status koha-common` para asegurar la correcta comunicación entre la cola de base de datos y Elasticsearch.

Resultados:

- Se confirmó la ejecución activa y estable del proceso `es_indexer_daemon.pl --batch_size 10`.
- Esto garantiza que cualquier alta o modificación bibliográfica en la instancia `bibliofadu` sea indexada e impacte en las búsquedas en un lapso de 5 a 10 segundos, eliminando la necesidad de reconstrucciones manuales del índice.
- Se detectó la ejecución paralela del demonio de Zebra (`zebrasrv`), lo cual se mantiene operativo exclusivamente para dar soporte a peticiones externas por protocolo Z39.50.

5. Implementación de Backups (Snapshots de Elasticsearch)

Para evitar la costosa reconstrucción de índices (158.000+ registros) en caso de corrupción, se implementó el sistema nativo de Snapshots incrementales.

Configuración del Repositorio:

1. Se creó el directorio `/var/backups/elasticsearch` con propiedad exclusiva para el usuario `elasticsearch`.
2. Se autorizó la ruta en `/etc/elasticsearch/elasticsearch.yml` mediante la directiva: `path.repo: ["/var/backups/elasticsearch"]`.
3. Se registró el repositorio lógico `koha_backup` vía API REST.

Política de Retención (15 días): Se implementó el script Bash `mantener_snapshots.sh` programado en el cron diario (03:30 AM). Este script genera una instantánea incremental diaria y utiliza la herramienta `jq` para purgar automáticamente aquellas con una antigüedad mayor a 15 días, optimizando el espacio en disco.

6. Herramientas de Monitoreo Creadas

Se dejó instalado y operativo el script `check_koha_es.sh` para auditorías rápidas por consola.

- Función: Consulta la API de Elasticsearch y cruza los datos con el comando `df` del sistema operativo.
- Salida: Muestra el estado de salud del clúster (Green/Yellow/Red), desglosa los índices pertenecientes a Koha (biblios, authorities), sus tamaños exactos en formato legible (KiB/MiB) y calcula el porcentaje real de ocupación de los índices respecto a la partición de almacenamiento.

Bitácora de Servidor: Generador Estático de Galería de Portadas (OPAC)

Servicio: Koha ILS

Instancia: bibliofadu

Componente: Interfaz OPAC - Galería de novedades

Archivo principal: `generate-gallery.php`

1. Objetivo del Script

El script tiene como finalidad consultar la base de datos de Koha para identificar los ingresos de material bibliográfico de los últimos 20 días, buscar sus respectivas portadas en servicios externos (APIs) y generar un fragmento HTML estático (`book_gallery.html`). Este HTML es luego consumido por el OPAC para mostrar un carrusel o grilla de novedades sin sobrecargar la base de datos ni el servidor web con consultas dinámicas recurrentes.

2. Refactorización y Mejoras Implementadas

Se realizó una reestructuración del código original para aplicar mejores prácticas de seguridad, rendimiento y tolerancia a fallos en el entorno de producción.

2.1. Seguridad y Configuración Dinámica

- Eliminación de credenciales en texto plano: Se modificó la conexión PDO a MariaDB/MySQL. El script ahora parsea dinámicamente el archivo `/etc/koha/sites/bibliofadu/koha-conf.xml` usando `simplexml_load_file()` para obtener el host, base de datos, usuario y contraseña. Esto evita la exposición de credenciales y previene rupturas si se cambia la clave del usuario de base de datos en el futuro.

2.2. Optimización de Base de Datos

- Compatibilidad con Strict Mode: Se corrigió la cláusula `GROUP BY` de la consulta SQL para incluir todos los campos no agregados en el `SELECT` (biblionumber, isbn, title). Esto previene errores de ejecución (`ONLY_FULL_GROUP_BY`) dependiendo de la configuración del motor de base de datos.
- Control de Memoria (LIMIT): Se agregó un límite (`LIMIT 50`) a la consulta SQL. En caso de una catalogación o importación masiva de registros, se evita cargar arrays gigantescos en la memoria de PHP, procesando solo el lote necesario para obtener los 6 libros finales requeridos para la galería.

2.3. Robustez en el Procesamiento de Datos

- Limpieza de metadatos MARC: Se implementó una expresión regular (`preg_match`) para sanitizar el campo ISBN proveniente de `biblioitems`. Esto filtra sufijos, notas o caracteres inválidos comunes en la catalogación y extrae estrictamente la cadena alfanumérica de 10 o 13 caracteres requerida por las APIs externas.

2.4. Tolerancia a Fallos y APIs Externas

- Manejo de Timeouts: Se reemplazó el uso básico de `file_get_contents()` por una función personalizada (`fetch_url_with_timeout()`) que utiliza `stream_context_create`. Esto impone un tiempo máximo de espera estricto (3-5 segundos) al consultar APIs, evitando que el script se bloquee indefinidamente si el servicio externo sufre una caída.
- Redundancia de Fuentes (Fallback): Se integró la API de Open Library como segunda opción de búsqueda. Si Google Books no devuelve resultados o falla, el script realiza una petición `HEAD` a Open Library y, de ser exitosa (HTTP 200), descarga la portada alternativa, aumentando significativamente la tasa de éxito para material regional o universitario.

2.5. Observabilidad (Logging)

- Trazabilidad: Se implementó una función temporal de registro de eventos (`log_portada()`). El script ahora documenta cada paso del proceso (extracción de ISBN, consultas a Google Books y Open Library, éxitos, fallos y timeouts) en un archivo de texto plano, facilitando la depuración y el monitoreo de los límites de las APIs.

3. Rutas y Archivos Involucrados

- Configuración leída: `/etc/koha/sites/bibliofadu/koha-conf.xml`
- Directorio de almacenamiento de imágenes: `/usr/share/koha/opac/htdocs/covers/`
- Archivo HTML generado: `/usr/share/koha/opac/htdocs/portada/book_gallery.html`
- Archivo de Log (Temporal): `/tmp/koha_portadas.log`

Bitácora de Servidor: Solución de Ataques Externos

Servidor: Catálogo Koha (catalogo.fadu.uba.ar) / Apache2 web server

Tema: Mitigación de escaneos automatizados y optimización de logs mediante Fail2Ban.

---

1. Descripción de la Incidencia

Se detectó un volumen inusual de entradas de error y advertencias en los logs de Apache y de la instancia de Koha (`/var/log/koha/bibliofadu/opac-error.log`). Los patrones identificados correspondían a:

- Peticiones HTTP/0.9 malformadas sin cabecera `Host` (forzando errores 400 en Plack/Koha).
- Escaneos ciegos buscando gestores de contenido inexistentes en esta arquitectura (ej. `wp-login.php`).
- Intentos de ataques de *Directory Traversal* manipulando URIs (`/cgi-bin/.%2e/.%2e/.../bin/sh`).

Diagnóstico: Tráfico generado por bots automatizados y escáneres de vulnerabilidades. No se detectaron brechas de seguridad, pero el "ruido" afectaba la legibilidad de los logs y consumía ciclos de procesamiento de red innecesarios.

2. Acciones Realizadas

Se decidió implementar un filtrado dinámico a nivel de firewall utilizando Fail2Ban para bloquear las IPs de origen de estas peticiones anómalas.

2.1. Creación de Filtro Personalizado

Se desarrolló un filtro de expresiones regulares (RegEx) específico para atrapar los tres patrones detectados sin generar falsos positivos con el tráfico legítimo de la biblioteca.

- Archivo creado: `/etc/fail2ban/filter.d/apache-koha-scanners.conf`

2.2. Configuración de la Jaula (Jail)

Se habilitó una nueva jaula para aplicar las penalizaciones, configurada explícitamente para leer archivos de texto plano, evitando problemas de rendimiento.

- Archivo creado: `/etc/fail2ban/jail.d/apache-koha-scanners.conf`
- Parámetros clave:

- `backend = auto`: Forzado localmente para sobrescribir la directiva global `systemd`. Esto evitó que Fail2Ban iterara innecesariamente sobre todo el *journal*, resolviendo la advertencia *"Jail started without 'journalmatch' set"* y manteniendo el consumo de CPU por debajo del 1%.
- `maxretry = 2` (en 10 minutos).
- `bantime = 86400` (baneo de 24 horas).
- `logpath`: Apuntando a los logs de error de Apache y a los específicos de la intranet y el OPAC de Koha.

2.3. Auditoría Remota vía Rsyslog

Para facilitar el monitoreo en tiempo real desde la estación de administración local (Arch Linux), se configuró el módulo `imfile` en el servidor. Esto permite capturar las líneas generadas en `/var/log/fail2ban.log` y reenviarlas a través del puerto 514 hacia el equipo de escritorio.

- Archivo creado: `/etc/rsyslog.d/90-fail2ban-forward.conf`

3. Pruebas y Validación

1. Validación del Filtro: Se ejecutó `fail2ban-regex` sobre el log histórico `opac-error.log`. El resultado arrojó 105 coincidencias exactas correspondientes a tráfico malicioso, ignorando correctamente las otras 428 líneas de eventos legítimos.
2. Validación de Rendimiento: Tras el inicio del servicio (`PID 2913552`), el consumo de recursos se estabilizó en niveles óptimos: CPU ~0.5% y RAM ~21.4M (RSS ~44MB).
3. Estado de Logs: Se validó que el mensaje *`WARNING Unable to retrieve Watch object`* originado por el módulo `filterpyinotify` es una respuesta normal a la rotación de logs de logrotate y no afecta la continuidad del monitoreo.

4. Estado Actual

El sistema se encuentra activo, filtrando escáneres web correctamente y optimizado para no generar carga en el servidor de base de datos ni en el servicio de catálogo de la universidad.

Bitácora de Servidor: Informe de Mantenimiento y Optimización de Base de Datos

Servicio: MariaDB Server (`50-server.cnf`)

Servicio: Catálogo Koha 23.05 - Biblioteca FADU (Debian GNU/Linux)

Objetivo: Tuning de rendimiento, eliminación de cuellos de botella por I/O de disco y liberación de memoria RAM.

1. Reasignación de Memoria RAM (InnoDB Buffer Pool)

- Diagnóstico: El parámetro `innodb_buffer_pool_size` estaba configurado en 10G, sobredimensionado para el tamaño real de la base de datos (menor a 1 GB).
- Acción: Se redujo el valor a 3G.
- Resultado: Se liberaron aproximadamente 10 GB de memoria RAM para el sistema operativo y otros servicios esenciales del catálogo. El `InnoDB Buffer Pool Hit Ratio` se mantiene estable en 100.0%.

2. Optimización de Tablas Temporales en Memoria

- Diagnóstico: Uso de valores por defecto para tablas temporales, lo que forzaba escrituras recurrentes en el disco físico.
- Acción: Se definieron explícitamente los límites `tmp_table_size = 128M` y `max_heap_table_size = 128M`.

3. Mitigación de I/O Físico por Tablas Temporales Complejas

- Diagnóstico: Las consultas complejas inherentes a la estructura de Koha (campos MARCXML/LONGTEXT) obligaban al motor a ignorar los límites de memoria y usar el disco físico.
- Acción: Se implementó un disco virtual en memoria RAM (`tmpfs`) de 2 GB. Se montó de forma persistente en `/var/lib/mysqltmp` (vía `/etc/fstab`) y se redirigió la variable `tmpdir` en la configuración de MariaDB.
- Resultado: Las escrituras forzadas ahora se realizan a la altísima velocidad de la RAM, eliminando el cuello de botella y la latencia del disco físico.

4. Ajuste de Caché de Hilos de Conexión (Thread Cache)

- Diagnóstico: El parámetro `thread_cache_size` se encontraba inactivo.

- Acción: Se activó y se fijó el límite en 128.
- Resultado: Tras 9 días de evaluación, el sistema registró un máximo de conexiones simultáneas (`Max_used_connections`) de 7. El manejo de la concurrencia es óptimo y el costo de procesamiento (CPU) para la gestión de hilos es virtualmente nulo.

5. Estado Final del Servicio

- Rendimiento de lecturas en RAM: 100%
- Memoria RAM recuperada y disponible: ~9.5 GB
- Cuellos de botella de disco: Resueltos satisfactoriamente.

Bitácora de Servidor: Resolución de Error 500 en OAI-PMH

Incidencia: El endpoint nativo OAI-PMH (`/cgi-bin/koha/oai.pl`) devolvía un Error 500 al solicitar el formato `oai_dc` (Dublin Core), mientras que el formato `marcxml` funcionaba correctamente.

Objetivo: Habilitar el formato nativo `marcxml` para la recolección desde el repositorio institucional EPrints, y homologar el formato `oai_dc` para los requerimientos del SNRD/MinCyT.

1. Diagnóstico y Análisis de Causa Raíz

El log de errores de Plack/Apache (`plack-error.log`) arrojaba una falla en el módulo de Perl `HTTP::OAI::Encapsulation.pm` al no encontrar el método `nodeType`.

Se detectó que la Preferencia del Sistema `OAI-PMH:ConfFile` estaba activada y apuntando a un archivo YAML externo (`/etc/koha/koha-oai.conf`). Al forzar a Koha a procesar el formato `oai_dc` mediante una hoja XSLT externa declarada en ese YAML, el módulo interno `XML::LibXSLT` devolvía una cadena de texto cruda (string) en lugar de un objeto DOM. Esto provocaba un pánico en el motor OAI de Perl, resultando en el Error 500.

2. Resolución Aplicada

Se determinó que la configuración a través de un archivo YAML externo era innecesaria y contraproducente, ya que Koha soporta la generación de metadatos `oai_dc` y `marcxml` de forma nativa e interna, sin depender de mapeos externos forzados.

Acciones tomadas:

1. En la interfaz de Intranet de Koha: Se vació por completo el valor de la Preferencia del Sistema `OAI-PMH:ConfFile`.
2. En la terminal del servidor: Se purgó la memoria caché para forzar la lectura de la nueva configuración mediante el comando: `systemctl restart memcached`

3. Estado Final

El motor OAI-PMH nativo de Koha ha retomado el control total de la diseminación de metadatos. El endpoint responde exitosamente (HTTP 200) a las peticiones en ambos formatos:

- MARCXML: `?verb=ListRecords&metadataPrefix=marcxml&set=tesis` (Para uso de EPrints).

- Dublin Core: `?verb=ListRecords&metadataPrefix=oai_dc&set=tesis` (Homologado para el SNRD).

Bitácora de Sysadmin: Koha + Elasticsearch + RabbitMQ

1. Gestión del Demonio de Elasticsearch (El Cartero)

Estos comandos manejan exclusivamente el indexador en tiempo real de Koha hacia ES.

- **Ver el estado del servicio:** `sudo koha-es-indexer --status bibliofadu`
- **Iniciar el servicio en segundo plano (Normal):** `sudo koha-es-indexer --start bibliofadu`
- **Detener el servicio:** `sudo koha-es-indexer --stop bibliofadu`
- **Correr el demonio en primer plano (Modo Debug/Rescate):** Útil para destrabar colas o ver errores en vivo en la terminal. `sudo koha-shell bibliofadu -c "perl /usr/share/koha/bin/workers/es_indexer_daemon.pl"`
- **Chequeo a nivel de Sistema Operativo (RAM):** Para verificar si el proceso está vivo o es un zombie, directo en la memoria del servidor. `ps aux | grep es_indexer_daemon | grep -v grep`

2. Auditoría de RabbitMQ (Colas de Mensajes)

Para verificar si los mensajes fluyen o si hay un embotellamiento.

- **Listar las colas activas y ver mensajes pendientes:** Si la cola `elastic_index` tiene un número mayor a 0 que no baja, el demonio está trabado.

  ```
  sudo rabbitmqctl list_queues | grep bibliofadu
  ```

- **Habilitar el protocolo STOMP (Requerido para el worker de tareas de Koha):**

  ```
  sudo rabbitmq-plugins enable rabbitmq_stomp
  ```

- **Purgar/Vaciar una cola manualmente (Avanzado):** Si una cola se llenó de miles de mensajes corruptos y querés vaciarla para empezar de cero sin reiniciar el servidor.

  ```
  sudo rabbitmqctl purge_queue koha_bibliofadu -elastic_index
  ```

3. Sincronización Forzada y Reconstrucción (Rebuild)

Cuando el indexador en tiempo real falló y necesitas empujar los datos a la fuerza desde MariaDB hacia Elasticsearch.

- **Reindexar TODO el catálogo:** `koha-elasticsearch --rebuild -d bibliofadu -v`
- **Reindexar UN SOLO registro específico:** Ideal para probar si un error de mapeo está rompiendo la carga. `koha-elasticsearch --rebuild -b <biblionumber> -d bibliofadu -v`

4. Diagnóstico Directo de Elasticsearch (Health Checks)

Consultas a la API nativa de ES (Puerto 9200) para ver la salud del motor, independientemente de Koha.

- **Estado general del Cluster:** Devuelve si el estado es `green`, `yellow` (faltan réplicas) o `red` (datos perdidos).

  ```
  curl -X GET "localhost:9200/_cluster/health?pretty"
  ```

- **Revisar uso de Memoria RAM (JVM) y Garbage Collector:** Para saber si ES se está ahogando o necesita más memoria en `jvm.options`.

  ```
  curl -X GET "localhost:9200/_nodes/stats/jvm?pretty"
  ```

- **Ver todos los índices, su peso y cantidad de documentos:** Te muestra una tabla rápida para comprobar si el índice de Koha está creciendo y cuánto pesa en disco.

  ```
  curl -X GET "localhost:9200/_cat/indices?v"
  ```

5. Reinicios de Servicios Auxiliares

Comandos para destrabar sockets o conexiones colgadas sin reiniciar todo el servidor.

- **Reiniciar la Interfaz Web (Plack):** Destraba errores de conexión entre la intranet y la cola de mensajes. `sudo koha-plack --restart bibliofadu`
- **Reiniciar el Worker de Tareas de Fondo:** Soluciona errores al procesar lotes o importaciones MARC. `sudo koha-worker --restart bibliofadu`
- **Reiniciar RabbitMQ:** `sudo systemctl restart rabbitmq-server`

INFORME TÉCNICO: INCIDENTE DE SEGURIDAD Y REESTRUCTURACIÓN DE ARQUITECTURA WEB

Fecha: 28 de Abril de 2026

Sistema: Repositorio y Catálogo Koha (FADU-UBA)

Afectación: Interfaz Pública (OPAC)

1. Antecedentes y Diagnóstico del Incidente

Durante la jornada se detectó una degradación significativa en los tiempos de respuesta del catálogo público (OPAC). El análisis de los registros del sist`ema` (htop y logs de acceso) evidenció un ataque de denegación de servicio de tipo "Scraping Lento y Distribuido".

- **Vector de ataque:** Múltiples direcciones IP (principalmente de origen ruso y europeo) ejecutaron peticiones secuenciales automatizadas mediante scripts en Python (`python-httpx/0.28.1`) dirigidas a los módulos pesados del sistema (`opac-search.pl`, `opac-detail.pl` y vistas MARC/ISBD).
- **Impacto en infraestructura:** El comportamiento de los bots forzó al servidor Apache (operando bajo el modelo `mpm_itk` por políticas de aislamiento del usuario `bibliofadu-koha`) a clonar procesos hasta alcanzar un límite crítico de 600 *workers* simultáneos, generando un embudo de procesamiento y riesgo de agotamiento de memoria RAM.

2. Mitigación Inmediata (Fase Reactiva)

Para contener la saturación sin afectar el trabajo del personal de biblioteca, se aplicaron las siguientes medidas de urgencia:

- **Filtrado por capa de aplicación:** Se establecieron reglas de reescritura en Apache (`RewriteRule`) para bloquear con código HTTP 403 el acceso a las vistas MARC en el entorno público, así como el bloqueo estricto de User-Agents correspondientes a herramientas de *scripting* (Python, Curl, Wget).
- **Fortalecimiento perimetral (Fail2Ban):** Se reconfiguró la "jaula" de protección ajustando el margen de tolerancia temporal (`findtime` = 300s) y límite de intentos (`maxretry` = 40) para detectar bots lentos. Se integró una lista blanca (`ignoreip`) para el segmento de red de la FADU (`157.92.122.0/23`), garantizando la operatividad ininterrumpida de los catalogadores en la Intranet.

- **Optimización de logs:** Se resolvió un solapamiento en las tareas programadas (Cron) de `awstats`, migrando la monitorización analítica a **GoAccess** en tiempo real mediante túneles WebSocket securizados y limitación de acceso por Autenticación Básica.

3. Implementación Arquitectónica Definitiva (Fase Proactiva)

Para evitar la recurrencia del agotamiento de procesos `mpm_itk`, se modificó la topología de red local del servidor pasando de un modelo de acceso directo a un modelo de Proxy Inverso:

- **Despliegue de Nginx en el Perímetro:** Se instaló y configuró Nginx en el puerto 443 como primer punto de contacto. Este motor asume la gestión y terminación del cifrado SSL/TLS, aliviando la carga criptográfica del servidor interno.
- **Limitación de Tasa (Rate Limiting):** Se implementó la directiva `limit_req_zone` en Nginx, estableciendo un umbral máximo de 3 peticiones por segundo por dirección IP (con ráfagas controladas). El tráfico malicioso es ahora rechazado instantáneamente en el borde (Error 503) sin consumir recursos de procesamiento de la base de datos MariaDB ni de la aplicación Koha.
- **Aislamiento de Apache:** El servidor Apache fue relegado a escuchar exclusivamente conexiones locales seguras (puerto 8080) provenientes de Nginx, recuperando su estabilidad y reduciendo su huella de memoria.

4. Integración de Inteligencia de Amenazas y Prevención de Intrusos (IPS)

Para complementar el límite de tasa estático configurado en el Proxy Inverso (Nginx) y la detección algorítmica local (Fail2Ban), se procedió a la instalación del motor de seguridad colaborativo **CrowdSec**.

- **Análisis de Comportamiento:** El motor fue integrado para la lectura bidireccional de los registros (logs) tanto de Nginx como de Apache, aplicando escenarios de mitigación que identifican ataques de Fuerza Bruta, Escaneo de Puertos y *Web Scraping* avanzado.
- **Sincronización Global (Threat Intelligence):** La principal ventaja arquitectónica implementada radica en la conexión del servidor con la base de datos global de CrowdSec. El sistema descarga y actualiza proactivamente listas de control de acceso (Blocklists) con direcciones IP identificadas en actividades maliciosas a nivel mundial.
- **Ejecución en Nivel Kernel:** La mitigación se hace efectiva a través del módulo `crowdsec-firewall-bouncer-iptables`, el cual inserta reglas de descarte directamente en el firewall del sistema operativo (Iptables/Netfilter). Esto asegura que el tráfico proveniente de las redes de bots (Botnets) sea neutralizado en la capa de red (Capa 3 del modelo OSI), garantizando un consumo nulo de recursos a nivel de aplicación web (Capa 7).

5. Centralización de la Interfaz Administrativa (Intranet) bajo Proxy Inverso

Para unificar la gestión de seguridad y simplificar la administración de certificados SSL, se integró la Intranet de Koha (puerto 8081) en la infraestructura de Proxy Inverso.

- **Encapsulamiento del Tráfico Administrativo:** Se reconfiguró el VirtualHost de la Intranet en Apache para aceptar únicamente conexiones locales (`127.0.0.1`), eliminando la exposición directa del puerto administrativo a redes externas.
- **Terminación SSL Unificada:** Nginx asume la responsabilidad del cifrado TLS para el puerto 8081. Esto permite gestionar las renovaciones de Let's Encrypt en un único punto, reduciendo la complejidad de la configuración de Apache.
- **Optimización de Sesiones de Catalogación:** Se ajustaron los parámetros de `proxy_read_timeout` y `proxy_connect_timeout` en el bloque de Nginx para soportar operaciones pesadas de importación de registros (MARC21) y ejecución de reportes SQL extensos, garantizando que el Proxy no corte conexiones legítimas durante procesos de larga duración.
- **Continuidad Operativa:** La arquitectura actual permite que el personal de biblioteca mantenga sus flujos de trabajo habituales de forma transparente, mientras se beneficia de la protección de capa 7 proporcionada por Nginx y el análisis de comportamiento de CrowdSec.

6. Aislamiento de Recursos y Alta Disponibilidad (QoS)

La convergencia del tráfico público (OPAC) y administrativo (Intranet) a través de un único Proxy Inverso (Nginx) fue diseñada garantizando el aislamiento de recursos y evitando puntos de contención (Cuellos de Botella).

- **Gestión Asíncrona:** A diferencia del modelo de concurrencia por procesos de Apache (que demostró vulnerabilidad al agotamiento de *workers*), Nginx procesa el tráfico mediante un bucle de eventos (Event Loop) no bloqueante. Esto le permite sostener miles de conexiones *Keep-Alive* de posibles atacantes con una huella de memoria (Footprint) insignificante, sin degradar el rendimiento general del sistema operativo.
- **Calidad de Servicio (QoS) Asimétrica:** La arquitectura establece políticas de limitación de tasa (`Rate Limiting`) de forma asimétrica. Las reglas de descarte restrictivas se aplican exclusivamente al VirtualHost del OPAC público. El tráfico dirigido al puerto 8081 (Intranet) fluye sin restricciones de tasa, garantizando prioridad absoluta de enrutamiento para el personal de la institución.
- **Separación de Procesos Backend:** La mitigación en el borde (Nginx) complementa la separación nativa de la aplicación Koha, donde los servicios de aceleración de Perl (Plack/Starman) operan como procesos del sistema (PID) independientes para el OPAC y la Intranet. La saturación teórica de un entorno no genera inanición de recursos (Resource Starvation) en el otro.

Bitácora de Infraestructura - Servidor: Hera (Catálogo Koha)

Fecha: 30 de Abril de 2026

Incidente: Ataque distribuido de extracción de datos (Scraping Botnet).

Patrón detectado: Picos de peticiones automatizadas desde múltiples IPs internacionales (Asia/África) apuntando a rutas pesadas de Koha (`/cgi-bin/koha/opac-search.pl`, `/opac-detail.pl`), concentrados en la franja horaria de 06:00 a 13:00 hs.

Acciones Tomadas y Estado:

1. **Validación de Rate-Limiting:** Se comprobó empíricamente que la directiva de "Sala de Espera Global" (límite de 15 peticiones por segundo en Nginx) implementada anteriormente salvó al servidor de la caída. Apache pudo procesar las peticiones encoladas manteniendo un *Load Average* óptimo.
2. **Filtro de Protocolo Obsoleto (Nginx):** Se detectó el uso de herramientas de scraping arcaicas. Se implementó una regla de bloqueo directo (`return 403;`) para cualquier conexión entrante usando el protocolo **HTTP/1.0**, fulminando la primera ola del ataque sin costo de CPU.
3. **Monitoreo de Mutación:** El ataque mutó hacia HTTP/1.1. Se decidió no bloquear para evitar "fuego amigo" a usuarios reales. El ataque fue contenido íntegramente por el rate-limiting de Nginx, manteniendo la carga de CPU en `0.05` (1% de uso real en los 4 núcleos).
4. **Auditoría de Backups (Koha a NAS):** Se revisó el pipeline de copias de seguridad. Quedó asentada la necesidad de mover la ejecución de `koha-run-backups` fuera de `/etc/cron.daily` hacia un horario explícito en la madrugada para evitar condiciones de carrera (Race Condition) con la tarea de `rclone` de las 07:00 AM.

Estado actual: Servidor estable, operando en modo "Búnker" y repeliendo ataques automatizados sin impacto en la experiencia de navegación de los alumnos y docentes.