RSA NETWITNESS® SUITE

Guía de ajuste de la base de datos de Core

para la versión 11.0

Información de contacto

RSA Link en https://community.rsa.com contiene una base de conocimientos que responde a las preguntas comunes y brinda soluciones para problemas conocidos, documentación de productos, análisis de la comunidad y administración de casos.

Marcas comerciales

Para obtener una lista de las marcas comerciales de RSA, visite mexico.emc.com/legal/emc-corporation-trademarks.htm#rsa (visite el sitio web de su país correspondiente).

Acuerdo de licencia

Este software y la documentación asociada son propiedad e información confidencial de EMC, se suministran bajo licencia, y pueden utilizarse y copiarse solamente de acuerdo con los términos de dicha licencia y con el aviso de copyright mencionado a continuación. No se puede suministrar a ninguna persona, ni poner a su disposición de cualquier otra manera, este software ni la documentación, o cualquier copia de estos elementos.

Este documento no constituye ninguna transferencia de titularidad ni propiedad del software, la documentación o cualquier derecho de propiedad intelectual. Cualquier uso o reproducción sin autorización de este software y de la documentación pueden estar sujetos a responsabilidad civil o penal.

Este software está sujeto a cambios sin aviso y no debe considerarse un compromiso asumido por EMC.

Licencias de otros fabricantes

Este producto puede incluir software que ha sido desarrollado por otros fabricantes. El texto de los acuerdos de licencia que se aplican al software de otros fabricantes en este producto puede encontrarse en la página de documentación del producto en RSA Link. Al usar este producto, el usuario acepta regirse totalmente por los términos de los acuerdos de licencia.

Nota sobre tecnologías de cifrado

Es posible que este producto contenga tecnologías de cifrado. Muchos países prohíben o limitan el uso, la importación o la exportación de las tecnologías de cifrado, y las regulaciones actuales de uso, importación y exportación deben cumplirse cuando se use, importe o exporte este producto.

Distribución

EMC considera que la información de esta publicación es precisa en el momento de su publicación. La información está sujeta a cambios sin previo aviso.

febrero 2018

Contenido

Presentación de la base de datos de NetWitness Core 7

Productos de NetWitness Suite incluidos en esta guía 7

Términos de uso frecuente 7

Historial de la base de datos de NetWitness Core 9

Fortalezas y debilidades de la base de datos principal 9

Configuración básica de la base de datos 11

Buscar ayuda dentro del servicio Core 11

Almacenamiento de paquetes, metadatos y sesiones 11

Almacenamiento del índice 11

Almacenamiento de la base de datos en niveles 12

Archiver 13

Manifiestos 14

Buscar manifiestos históricos 15

Configuración avanzada de la base de datos 17

Nodos de configuración de la base de datos 17

packet.dir , meta.dir , session.dir 17

packet.dir.warm , meta.dir.warm , session.dir.warm 18

packet.dir.cold , meta.dir.cold , session.dir.cold 19

packet.file.size , meta.file.size , session.file.size 19

packet.files , meta.files , session.files 20

packet.free.space.min , meta.free.space.min , session.free.space.min 20

packet.index.fidelity , meta.index.fidelity 20

packet.integrity.flush , meta.integrity.flush , session.integrity.flush 21

packet.write.block.size , meta.write.block.size , session.write.block.size 21

packet.compression , meta.compression 21

packet.compression.level , meta.compression.level 22

hash.algorithm 22

hash.databases 22

hash.dir 22

Nodos de configuración del índice 22

index.dir 23

index.dir.warm 23
index.dir.cold 23
index.slices.open 23
page.compression 24
save.session.count 24
reindex.enable 24
Nodos de configuración de SDK 24
max.concurrent.queries 24
max.pending.queries 25
cache.window.minutes 25
max.where.clause.cache 25
max.unique.values 25
query.level.1.minutes , query.level.2.minutes , query.level.3.minutes 25
query.timeout 26
max.where.clause.sessions 26
max.query.groups 26
packet.read.throttle 27
cache.dir , cache.size 27
parallel.values 27
parallel.query 27
Nodos de configuración por usuario 28
query.prefix 28
query.level 28
query.timeout 28
session.threshold 28
Programador 29
Ejemplo 29
Rollover 30
Transferencia síncrona 30
Transferencia asíncrona 30
Ejemplo 31

Consultas 33

query Sintaxis 33
where Cláusulas 35
Operadores de consulta 36
Valores de texto 37

Direcciones IP 37
Direcciones MAC 37
Expresiones de fecha y hora 37
Puntos de tiempo relativos 38
Valores de rango especiales 38
Cláusula group by (desde 10.5) 39
Cláusula order by (desde 10.5) 41
Llamada values 42
Parámetros 43
Marcas de values 45
Ejemplo de llamada values 45
Llamada msearch 46
Marcas de msearch 47
Modo de búsqueda de índice de msearch 47
Consejos para msearch 48
Procedimientos almacenados 48
Uso de comillas en la sintaxis de una consulta 48

Personalización del índice 51

Ubicaciones del archivo de configuración del índice 51
Entradas de configuración del índice 51
Nombres de metadatos 52
Tipos de datos 52
Niveles de índice 53
Valor máximo 54
maxLength 54
Cambio de nombre de clave 55

Reconstrucción del índice 57

Activación del reindexador en segundo plano 57
Control del reindexador en segundo plano 57
Algoritmo de reindexación en segundo plano 57
Estado del indexador en segundo plano 58
Efectos en la agregación 58
Cómo forzar una reindexación 58

Técnicas de optimización 59

Límites 59

Cláusulas where complejas 59
AND y OR 60
Caso de uso: Coincidencia con una subred grande 60
Caso de uso: Coincidencias de subcadena 61
Guardados del índice 62
Efectos de aumentar el intervalo de guardado 62
Efectos de disminuir el intervalo de guardado 63
Trabajo con valueMax 63
Paralelizar cargas de trabajo 63
Reconstruir el índice 64
Escalamiento de retención 64
Aumentar la retención de paquetes y metadatos 64
Aumentar la retención del índice 65
Escalar de manera horizontal 65
Cargas de trabajo de grupos 65
Ventana de caché 66
Límites de tiempo 67

Apéndice A: Estadísticas 69

Estadísticas en /database/stats 69
Estadísticas en /index/stats 70
Estadísticas en /sdk/stats 71
Estadísticas por consulta 71

Apéndice B: Inspección del índice 73

Parámetros 73
Respuesta 73
Resumen del segmento 73
Resumen por índice 73
Pie de página del resumen del segmento 74

Presentación de la base de datos de NetWitness Core

En este tema se proporciona una descripción general de la base de datos de NetWitness Core. Los servicios NetWitness Core contienen una base de datos de propiedad desarrollada específicamente para su uso dentro de los productos NetWitness Suite. Se parece poco a las bases de datos relacionales tradicionales y no se basa en ninguna tecnología de base de datos disponible para la venta. Como tal, muchos usuarios encuentran que hay una curva de aprendizaje empinada para entender cómo funciona la base de datos Core y cómo utilizarla de la mejor manera. El propósito de esta guía es ayudar a los usuarios de NetWitness Suite a entender la base de datos y utilizarla a su potencial máximo.

Como administrador del sistema, puede utilizar esta información como ayuda para planear la implementación de NetWitness Suite y ajustarla para obtener el mejor rendimiento. Como analista, puede utilizar esta guía para estructurar su análisis de manera que se obtengan informes más rápido. Como desarrollador de contenido, puede utilizar esta guía como ayuda para redactar contenido que procesará eficazmente el sistema de base de datos.

Productos de NetWitness Suite incluidos en esta guía

En esta guía se hace referencia a las funcionalidades de NetWitness Suite 11.0. Los siguientes componentes de NetWitness Suite contienen la base de datos principal:

- Concentrator
- Archiver
- Decoder
- Log Decoder
- Workbench

Términos de uso frecuente

Aquí se presentan las definiciones de términos que se utilizan en este documento. Los términos se enumeran en el orden de ingreso en el sistema NetWitness Suite:

- **Base de datos de paquete** : la base de datos de paquete contiene los datos crudos capturados. En un Decoder, la base de datos de paquete contiene paquetes que se capturan de la red. Los Log Decoders utilizan la base de datos de paquete para almacenar registros crudos. Es

posible acceder a los datos crudos almacenados en la base de datos de paquete mediante un ID de paquete; sin embargo, generalmente este ID nunca es visible para el usuario final.

- **ID de paquete** : Número que se usa para identificar de manera exclusiva un paquete o un registro en una base de datos de paquete.
- **Base de datos de metadatos** : la base de datos de metadatos contiene información que un Decoder o Log Decoder extrae del flujo de datos crudos. Los analizadores, las reglas o los feeds pueden generar elementos de metadatos.
- **ID de metadatos** : Número que se usa para identificar de manera exclusiva un elemento de metadatos en la base de datos de metadatos.
- **Clave de metadatos** : nombre que se utiliza para clasificar el tipo de cada elemento de metadatos. Las claves de metadatos comunes incluyen ip.src, time o service.
- **Valor de metadatos** : cada elemento de metadatos contiene un valor. El valor es lo que genera cada analizador, feed o regla.
- **Base de datos de sesión** : La base de datos de sesión contiene información que vincula el paquete y los elementos de metadatos juntos en sesiones.
- **Sesión** : en un Decoder de paquete, una sesión representa un solo flujo de red lógica. Por ejemplo, una conexión TCP/IP es una sesión. En un Log Decoder, cada evento de un registro es una sesión. Cada sesión contiene las referencias a todos los ID de paquete e ID de metadatos que hacen referencia a la sesión.
- **ID de sesión** : Número que se usa para identificar de manera exclusiva a las sesiones en la base de datos de sesión.
- **Índice** : El índice es una recopilación de archivos que proporciona una forma de buscar ID de sesión con el uso de valores de metadatos.
- **Base de datos Core** : Se refiere a la combinación de paquete, metadatos, sesión e índice.

En el caso de las definiciones de sintaxis, este documento utiliza definiciones gramaticales de EBNF .

Historial de la base de datos de NetWitness Core

NetWitness (ahora RSA) desarrolló la base de datos Core para su uso en sistemas de captura de paquetes. Al principio de la historia de NetWitness, los desarrolladores identificaron que las tecnologías de bases de datos existentes no serían capaces de mantenerse al día con la alta tasa de recopilación inherente a la captura de paquetes completa. Las tecnologías de bases de datos contemporáneas no estaban ni cerca de ser capaces de mantenerse al día con la captura del número de sesiones recibido cada segundo, y mucho menos con la clasificación de cada paquete. Del mismo modo, el volumen de datos significaba que sería necesario desechar el almacenamiento de paquetes y reutilizarlo tan rápidamente como se consumiera. Esta fue también una debilidad de las bases de datos en el momento. De este modo, NetWitness creó una base de datos que incluye el paquete, la sesión y bases de datos de metadatos.

Con el fin de proporcionar las funcionalidades analíticas de NetWitness Investigator, se añadió un índice de metadatos a la base de datos de NetWitness. El índice comparte los mismos objetivos de diseño que las bases de datos originales. Se diseñó para sostener una tasa muy alta de inserción en un gran número de índices muy grandes.

El índice ha evolucionado considerablemente en los últimos años. Las primeras versiones del índice solo fueron capaces de proporcionar estimaciones resumidas acerca de cómo muchos valores de metadatos únicos estaban presentes en la base de datos de metadatos. Otras versiones han tenido grandes dificultades para alcanzar el rendimiento de consulta aceptable. Por ejemplo, NetWitness 9.0 medía con más frecuencia los tiempos de informes en minutos en lugar de segundos. La versión actual del índice se deriva del índice de NetWitness 9.0, pero ha evolucionado considerablemente para cumplir con las expectativas de rendimiento y agregar funciones nuevas.

Fortalezas y debilidades de la base de datos principal

Fortalezas:

- Tasas de inserción altas y sostenidas, sin necesidad de tiempo de inactividad para inserciones en masa.
- Rendimiento de consultas aceptable simultáneo con altas tasas de inserción.
- Limpieza automática y transferencia de datos antiguos con fragmentación mínima.
- Número de índices de valor de metadatos extremadamente alto: más de 100 activados de forma predeterminada en un Concentrator.
- Capacidad de escalar a tamaños de bases de datos en petabytes y tamaños de índice en terabytes dentro de un solo nodo.

- Cuando se usan pares de valores clave de metadatos es muy flexible para almacenar elementos de metadatos arbitrarios dentro de una sesión. Así, una sesión puede usarse para representar casi cualquier tipo de registro de datos.

Debilidades:

- La funcionalidad de consulta es limitada y de bajo nivel.
- El esquema de paquetes, metadatos y base de datos de sesión es fijo, y toda la personalización se realiza mediante claves y valores de metadatos personalizados.
- La base de datos no ofrece garantías de atomicidad de transacción como se podría esperar encontrar en una base de datos SQL.

Configuración básica de la base de datos

En este tema se abordan los ajustes de configuración básicos de la base de datos de los servicios NetWitness Core. Para obtener información sobre cómo configurar los servicios principales mediante la edición de archivos de configuración, consulte “Ajustes de configuración de servicios” en la *Guía de introducción de hosts y servicios* .

Este documento supone que el lector tiene cierta familiaridad con el ajuste de la configuración de un servicio NetWitness Core. Para usarlo, debe estar familiarizado con uno de los mecanismos que permiten modificar el árbol de configuración de los servicios Core. Ejemplos de tales mecanismos incluyen la vista Explorador de las páginas Administration dentro de la interfaz del usuario de NetWitness Suite o la interfaz de REST, a la cual se accede en cada servicio a través de un navegador web.

Buscar ayuda dentro del servicio Core

Cada elemento de configuración dentro de un servicio Core tiene una descripción de la ayuda incorporada de la función del elemento. Puede ver esta información de ayuda si mantiene el mouse sobre el elemento de configuración en la vista Explorador. Cada elemento de configuración también indica si se puede cambiar sin necesidad de un reinicio o si es necesario reiniciar el servicio para que se aplique el cambio.

Los desarrolladores que usan la API REST pueden recuperar el texto de ayuda de cada elemento de configuración mediante el envío del mensaje `help` a la ruta del nodo de configuración.

Almacenamiento de paquetes, metadatos y sesiones

Cada una de las bases de datos de paquetes, metadatos y sesiones se configura a través de la carpeta `/database/config` en cada servicio NetWitness Core. Cada base de datos tiene un parámetro configurable para especificar el lugar donde el servicio Core almacena los datos. Las bases de datos de paquetes, metadatos y sesiones siguen un patrón predecible para todas sus entradas de configuración. Los elementos de configuración de la base de datos de paquetes se inician con el prefijo `packet` , la configuración de la base de datos de metadatos, con el prefijo `meta` y los elementos de configuración de la base de datos de sesiones, con el prefijo `session` .

Almacenamiento del índice

La configuración del índice se almacena en la carpeta `/index/config` en cada servicio principal.

Temas

- Almacenamiento de la base de datos en niveles
- Manifiestos

Almacenamiento de la base de datos en niveles

En este tema se describe el almacenamiento de base de datos en niveles y se proporcionan recomendaciones para el almacenamiento en niveles activo, semiactivo e inactivo.

A partir de la versión 10.4, el servicio Archiver se puede configurar para el uso de almacenamiento en niveles. El concepto de almacenamiento en niveles consiste en colocar los datos más recientes en un nivel activo, que es el almacenamiento más rápido disponible en Archiver.

De forma predeterminada, todos los servicios usan el nivel activo.

El nivel siguiente se conoce como semiactivo y suele ser almacenamiento más económico y más lento, como el almacenamiento conectado en red (NAS). El nivel semiactivo contiene datos más antiguos; la antigüedad depende del volumen de almacenamiento que se asigna en el nivel activo y de la tasa recopilación promedio. Cuando el nivel activo alcanza la utilización máxima, la progresión natural es transferir los datos más antiguos del nivel activo al nivel semiactivo. Cuando se configura correctamente, esto sucede de manera automática y es invisible para el usuario final. Las consultas y el acceso a los datos suceden automáticamente sin importar el nivel (activo o semiactivo) en el cual residen los datos. Sin embargo, puede haber un impacto en el rendimiento cuando se accede a los datos en el nivel semiactivo en comparación con el nivel activo, debido a que los tiempos de acceso en el nivel semiactivo suelen ser más lentos.

Además de los niveles activo y semiactivo, también hay un nivel inactivo. El nivel inactivo solo se usa como portapapeles para el respaldo offline. Los servicios NetWitness Core no acceden a datos en el nivel inactivo. Estos servicios transfieren los datos más antiguos al nivel inactivo y los consideran abandonados (el servicio ya no accede a ellos). Posteriormente, estos datos se pueden respaldar en almacenamiento a largo plazo, como cinta, para una posible restauración meses o incluso años después, en función de los requisitos. El respaldo y la posterior eliminación de los datos en el nivel inactivo se deben manejar fuera de los servicios NetWitness Core mediante scripts u otros procesos.

Si el nivel inactivo se llena debido a que los procesos externos no están quitando los datos a tiempo, el servicio NetWitness Core detiene finalmente la recopilación de nuevos datos hasta que se resuelve el problema.

Cuando los datos se transfieren al nivel inactivo, RSA recomienda que el directorio permanezca en el mismo punto de montaje de origen de la transferencia. Por lo tanto, si los archivos provienen del nivel semiactivo, es mucho mejor, por motivos de rendimiento, configurar el directorio del nivel inactivo en el mismo sistema de archivos. La razón de esto es que el servicio intenta transferir simplemente el archivo y el directorio al nivel inactivo, lo cual es una operación casi instantánea en el mismo sistema de archivos. Si la transferencia falla, el último recurso es copiar los datos al nivel inactivo, lo cual conlleva un mayor tiempo de procesamiento y causa una contención adicional de I/O en el nivel de origen de la copia.

Archiver

Archiver usa los niveles de funcionalidades de almacenamiento. Puede configurar Archiver de modo que use solo el almacenamiento activo (el valor predeterminado), el almacenamiento activo y semiactivo o los tres (activo, semiactivo e inactivo). Todos los servicios deben usar el almacenamiento activo. No puede configurar un servicio para que solo use el almacenamiento semiactivo. Los datos fluyen desde el almacenamiento activo al semiactivo y, finalmente, al inactivo. También puede omitir el almacenamiento semiactivo y pasar del activo al inactivo. Si el almacenamiento inactivo (offline) no está configurado, los datos más antiguos se eliminan en el último nivel configurado, lo que ha sido el procedimiento operativo estándar.

La implementación típica de Archiver configura todas las bases de datos en un tamaño ilimitado (packet.dir, meta.dir, session.dir, index.dir y, opcionalmente, las variantes del nivel semiactivo), lo cual significa que el especificador de tamaño se deja desactivado o se configura en cero. Esto permite que las bases de datos y el índice crezcan sin límites. En lugar de que cada base de datos administre su propio tamaño y se implemente solo cuando cada una supere su tamaño configurado, Archiver implementa todos los elementos juntos mediante el comando `/index sizeRoll` . Esto permite que las bases de datos y el índice se implementen simultáneamente. Para obtener más información sobre sizeRoll, consulte “Transferencia asíncrona” en Transferencia .

Generalmente, Archiver está configurado para colocar las bases de datos de índice, sesión, metadatos y paquetes (registro) en el mismo volumen en lugar de múltiples volúmenes, como sucede con Concentrator o Decoder. Aunque esto puede causar una mayor contención de I/O cuando se producen lecturas simultáneas en múltiples bases de datos, también maximiza la retención general. Debido a que todas las bases de datos están en el mismo volumen, están configuradas para implementarse juntas, lo cual minimiza la orfandad de los datos. Decoder y Concentrator están configurados para una velocidad máxima de I/O, pero puede ser difícil calcular el dimensionamiento correcto de los volúmenes.

Por ejemplo, si la base de datos de sesión es demasiado grande, puede tener almacenamiento suficiente para seis meses de retención, mientras que la base de datos de metadatos y el índice solo tienen retención para cuatro meses. Debido a que la base de datos de sesión y de metadatos y el índice están intrínsecamente vinculados, el período de retención más breve para los tres define el período de retención general (en este caso, cuatro meses). La retención de bases de datos individuales se ve afectada principalmente por factores que escapan a nuestro control, como el tráfico capturado, los metadatos generados (analizadores, feeds y reglas) y el filtrado. Las bases de datos se redimensionan fácilmente con un simple cambio en la configuración, pero esto suele implicar cambios en el nivel de hardware y del sistema de archivos para ajustar las particiones, lo cual complica el redimensionamiento dinámico. Archiver evita estos problemas mediante el uso de un único volumen para todo, con la desventaja de una velocidad de I/O algo más lenta.

Manifiestos

En este tema se describen los archivos de manifiesto y se proporciona un ejemplo de manifiesto para un archivo de base de datos de metadatos. También se describe la búsqueda de manifiestos y se proporciona un ejemplo de búsqueda manifiesto.

Los archivos de manifiesto se crean con cada archivo de base de datos de sesión, metadatos y paquete (registro) y con cada directorio de segmento de índice. Un archivo de manifiesto es un archivo que describe varias piezas clave de información acerca de los datos a los que hace referencia. Los archivos de manifiesto se escriben como un registro JSON. Los archivos de manifiesto transportan entre niveles los datos que representan. Si los datos que representan se eliminan, el archivo de manifiesto también se elimina, excepto en el siguiente caso especial. Si el servicio tiene `/database/config/manifest.dir` configurado en un directorio válido, en el momento en que se eliminan los datos del manifiesto, una copia del archivo de manifiesto se coloca en el directorio que indica `manifest.dir` (si el directorio no existe, se crea). Esto permite una función de NetWitness Suite denominada búsqueda de manifiestos históricos.

La intención de este proceso es mantener los archivos de manifiesto históricos durante años, en una ubicación para consulta offline. Como se puede imaginar de un servicio que se ejecuta durante muchos años, potencialmente esto puede generar cientos de miles de archivos. Esto no debería ser una preocupación, sin embargo, el servicio comprime automáticamente los archivos en un único archivo con el fin de ahorrar espacio cuando crecen demasiado. Los archivos de manifiesto son muy pequeños y se comprimen bien.

Ejemplo de manifiesto (`meta-000000023.nwmdb.manifest`) de un archivo de base de datos de metadatos:

```
{
    "filename" : "meta-000000023.nwmdb",
    "size" : 185153768,
    "fileTime" : 1403903940,
```

```
    "id1" : 150814110,
    "id2" : 159341086,
    "session1" : 4023382,
    "session2" : 4250442,
    "time1" : 1403903879,
    "time2" : 1404739851
}

filename  = The filename for the db file the manifest represents
size      = The size in bytes of the db file
fileTime  = The time the file was created
id1       = The starting id in the file (for this example, the starting
meta ID)
id2       = The last id in the file (for this example, the last meta ID)
session1  = The starting session ID of the first meta in the file
session2  = The last session ID of the last meta in the file
time1     = The POSIX time of the first "time" meta found in the file
time2     = The POSIX time of the last "time" meta found in the file
```

En este ejemplo de manifiesto, los campos más importantes son `fileTime` , `time1` y `time2` . Los tres campos se escriben en hora de POSIX. `time1` y `time2` corresponden a la hora de inicio y detención de los metadatos registrados en el archivo de base de datos de metadatos `meta-000000023.nwmdb` . Específicamente, `fileTime` siempre es la hora en que se creó el archivo (no cuando se modificó por última vez). `time1` y `time2` representan los rangos mínimo y máximo de los datos analizados dentro del archivo de base de datos de metadatos. Cuando se realizan búsquedas históricas por hora, `time1` y `time2` se prefieren sobre `fileTime` , cuando están presentes. Los archivos de manifiesto para las otras bases de datos y el índice contienen algunos campos diferentes, pero todos tienen suficiente información para ejecutar consultas basadas en hora.

Buscar manifiestos históricos

Cuando los manifiestos se recopilan en el directorio que señala `manifest.dir` , se da por hecho que los datos a los que se refieren se copiaron al nivel inactivo y se respaldaron finalmente en el almacenamiento offline. Debido a que el servicio mantiene el acceso a los manifiestos históricos, esto permite ejecutar consultas basadas en hora en datos offline con el fin de determinar los datos que se deben restaurar para un rango de tiempo dado.

Puede buscar manifiestos con el comando `/database manifest` :

```
manifest: If a manifest directory is defined, it will allow operations
on the manifest files (such as a time based query) for database files in
cold storage.
security.roles: database.manage
parameters:
   op - <string, optional, {enum-one:query|compress}> The operation to
perform (defaults to query)
   time1 - <date-time, optional> The beginning time (UTC) for matching
offline database files
   time2 - <date-time, optional> The ending time (UTC) for matching
offline database files
   timeFormat - <string, optional, {enum-one:posix|simple}> Specify the
time format that is returned (posix, simple), default is posix
```

Ejemplo de búsqueda:

```
/database manifest time1="2014-04-20 11:00:00" time2="2014-04-11
11:20:00" timeFormat=simple
```

La búsqueda devuelve todos los manifiestos que coinciden con la consulta:

```
[ filename=meta-000001691.nwmdb size=4843826176 fileTime="2014-Apr-20
11:06:34" id1=301555027452 id2=301733101896 session1=15352020201
session2=15361024200 time1="2014-Apr-20 11:05:34" time2="2014-Apr-20
11:16:34" compression=gzip ]
[ filename=session-000001865.nwsdb size=268439552 fileTime="2014-Apr-20
11:06:35" id1=14674145801 id2=14682041000 metaId1=288217522208
metaId2=288370660984 packetId1=11733872441 packetId2=11741745303 ]
[ filename=session-000001866.nwsdb size=268439552 fileTime="2014-Apr-20
11:18:31" id1=14682041001 id2=14689936200 metaId1=288370660985
metaId2=288520616949 packetId1=11741745304 packetId2=11749618589 ]
```

Los resultados devueltos se pueden utilizar para correlacionar los archivos que deben restaurarse desde el respaldo para el rango de tiempo dado. En el caso de NetWitness Suite 10.4 y superior, se puede usar un servicio denominado Workbench para tomar los archivos restaurados y proporcionar una interfaz de consulta de los datos restaurados mediante una o más recopilaciones.

La configuración del servicio Workbench escapa del alcance de este documento. Para obtener más información, consulte “Configurar el respaldo y la restauración de datos” en la *Guía de configuración de Archiver* .

Configuración avanzada de la base de datos

En este tema se explican las opciones de configuración avanzadas de la base de datos de NetWitness Core.

Las opciones de configuración de la base de datos de NetWitness Core pueden cambiar entre versiones. Sin embargo, muchos de los elementos de configuración no cambian con frecuencia y están documentados aquí. Esta no es una lista exhaustiva, ya que en cada versión se agregan nuevas funciones, las cuales pueden requerir nuevos elementos de configuración. Para obtener la documentación más actualizada, consulte la funcionalidad de ayuda incorporada del servicio NetWitness Core.

Temas

- Nodos de configuración de la base de datos
- Nodos de configuración del índice
- Nodos de configuración de SDK
- Nodos de configuración por usuario
- Programador
- Transferencia

Nodos de configuración de la base de datos

En este tema se describen los nodos de configuración de la base de datos. Los siguientes nodos de configuración de base de datos son algunos de los elementos de configuración avanzados de la base de datos de NetWitness Core que no cambian con frecuencia.

`packet.dir,meta.dir,session.dir`

Esta es la entrada de configuración principal de cada base de datos (también se conoce como nivel activo). Controla en qué parte del sistema de archivos se almacenan las respectivas bases de datos. Esta entrada de configuración comprende una sintaxis compleja para especificar muchos directorios como ubicaciones de almacenamiento.

Sintaxis de configuración:

```
config-value = directory, { ";" , directory } ;
directory    = path, [ ( "=" | "==" ) , size ] ;
path         = ? linux filesystem path ? ;
size         = number size_unit ;
```

```
size_unit     = "t" | "TB" | "g" | "GB" | "m" | "MB" ;
number        = ? decimal number ? ;
```

Ejemplo:

```
/var/lib/netwitness/decoder/packetdb=10
t;/var/lib/netwitness/decoder0/packetdb=20.5 t
```

Los valores de tamaño son opcionales: Si están configurados, indican el tamaño máximo total de los archivos almacenados ahí antes de que se realice una transferencia en las bases de datos. Si el tamaño no está presente, no se realiza una transferencia automática en la base de datos, pero el tamaño puede administrarse a través de otros mecanismos.

El uso de = o == es significativo. El comportamiento predeterminado de las bases de datos es crear automáticamente directorios especificados cuando se inicia el servicio Core. Sin embargo, este comportamiento se puede reemplazar con el uso de la sintaxis == . Si se usa == , el servicio no crea ningún directorio. Si no existen los directorios cuando se inicia el servicio, el servicio no comienza correctamente el procesamiento. Esto da al servicio resistencia contra los sistemas de archivos que faltan o que no están montados cuando arranca el host.

Si modifica el tamaño de un directorio en uso, el tamaño surte efecto inmediatamente, siempre que sea mayor. Se omite si es 10 % menor que el tamaño existente. Esto evita un error de escritura accidental que puede ocasionar una enorme pérdida de datos. Por ejemplo, si la base de datos de paquetes se configuró para `12 TB` y alguien la ingresó erróneamente como `12 GB` , la base de datos podría llegar a eliminar más de 11 TB de datos para reducirla a solo 12 GB. En lugar de esto, la base de datos no hace caso del ajuste de `12 GB` y registra una advertencia, de modo que el error se pueda detectar rápidamente. Por supuesto, si el tamaño especificado es realmente correcto y se diferencia en más de un 10 % del tamaño existente, el único recurso para que surta efecto es reiniciar el servicio. Cuando inicia el respaldo, supone que el tamaño es correcto y ajusta la base de datos al tamaño nuevo mediante la implementación de los datos más antiguos hasta alcanzar el tamaño nuevo. Si realmente quiere ajustar el tamaño a un tamaño menor y en más de un 10 % sin necesidad de reiniciar el servicio, debe modificar el tamaño varias veces mediante un ajuste de menos del 10 %, cada vez. Observe los registros del servicio para saber cuándo la base de datos se ha adaptado al nuevo tamaño, ya que solo ajusta el tamaño total de la base de datos cuando se cierra el último archivo que se escribe.

Si se agregan o eliminan directorios nuevos (separados por punto y coma), no se aplican sino hasta que se reinicia el servicio.

packet.dir.warm,meta.dir.warm,session.dir.warm

Estos ajustes son opcionales y se usan para el almacenamiento de nivel semiactivo en Archiver. De forma predeterminada, están en blanco y no se usan. Si se configuran, tienen el mismo formato y comportamiento que `packet.dir` , `meta.dir` y `session.dir` (consulte _ `packet.dir` , `meta.dir` y `session.dir` _ anteriormente). Una vez que se configuran, el archivo más antiguo del nivel activo se transfiere al nivel semiactivo cuando no queda espacio disponible en el nivel activo.

packet.dir.cold,meta.dir.cold,session.dir.cold

Estos ajustes son opcionales y se usan para transferir archivos de un sistema de almacenamiento de nivel activo o semiactivo al directorio de nivel inactivo que se especifica. Específicamente, este ajuste no es más que un directorio, no hay especificadores de tamaño. Sin embargo, el nombre de ruta definido tiene algunos especificadores de formato especiales que puede utilizar para nombrar el directorio con la fecha de los datos en él.

```
%y   = The year of the data being moved to the cold tier
%m   = The month of the data being moved to the cold tier
%d   = The day of the data being moved to the cold tier
%h   = The hour of the data being moved to the cold tier
%##r = A block of time within a day. So %12r would create two blocks, 00
and 01\. 00 for all data in the AM, 01 for all PM data
```

Ejemplo de ajuste:

```
packet.dir.cold = /var/lib/netwitness/archiver/database1/alldata/cold-
storage-%y-%m-%d-%8r
```

En el caso del ajuste anterior, si un archivo de la base de datos de registros estaba a punto de transferirse al almacenamiento inactivo y se creó el `2014-03-02 15:00:00` , se transferiría al siguiente directorio del nivel inactivo:

```
/var/lib/netwitness/archiver/database1/alldata/cold-storage-2014-03-02-
01
```

El último número `01` necesita explicación. El especificador `%8r` divide las horas del día en 24 / 8 = 3 partes. Las primeras ocho horas del día serían el bloque `00` , de las 00:00 h a las 08:00 h. Las ocho horas siguientes son entre las 08:00 h y las 16:00 h, y se les asigna el bloque `01` . Puesto que los datos que se transfieren al almacenamiento inactivo se crearon a las 15:00 h, caen dentro del bloque `01` . El especificador de formato `%r` es útil para respaldar archivos con una granularidad en algún punto entre un día `%d` y una única hora `%h` . El directorio de almacenamiento inactivo se crea según demanda y lo definen los datos que se transfieren cuando se usan especificadores de formato.

La capacidad de agregar una fecha a la ruta de los datos es solo una comodidad que se agrega para respaldo y restauración. Es una manera de etiquetar los datos con una fecha en la ruta.

packet.file.size,meta.file.size,session.file.size

Controla el tamaño de los archivos que se crean con cada base de datos. Generalmente no es necesario cambiar estos valores porque los valores predeterminado habitualmente funcionan bien. Este ajuste se aplica de inmediato para los archivos posteriores.

packet.files,meta.files,session.files

Este ajuste controla la cantidad de archivos que la base de datos mantiene abiertos. Puede aumentar este valor para mejorar el rendimiento: sin embargo, el sistema operativo tiene un límite global de la cantidad de archivos que el servicio puede mantener abiertos. Si se excede el límite, se informa un error y el servicio no funciona. Este ajuste se aplica de inmediato.

En NetWitness Suite 10.6 y superior, el valor predeterminado para packet.files, meta.files y session.files es `auto` y el servicio administra la cantidad de archivos abiertos en función de estos criterios:

1. Cantidad de recopilaciones
2. Cantidad de memoria del sistema

Cuando se configura en `auto` , el número es dinámico y se puede ver en los registros cuando cambia. Para NetWitness Suite 10.6, RSA recomienda configurar este valor en `auto` y no cambiarlo a un número específico.

packet.free.space.min,meta.free.space.min, session.free.space.min

Este ajuste proporciona un límite de seguridad en el espacio libre mínimo que existe en las rutas especificadas por los directorios packet.dir, meta.dir y session.dir, respectivamente. Este ajuste se usa para evitar que el servicio se quede sin espacio en el caso de que otros programas hayan llenado el espacio que debería estar dedicado a cada una de las bases de datos. Este ajuste se aplica de inmediato.

packet.index.fidelity,meta.index.fidelity

Este ajuste controla la frecuencia con se indexan ubicaciones de ID de paquete y ubicaciones de ID de metadatos. Este ajuste se puede aumentar para reducir la cantidad de espacio que necesita cada paquete o archivo de metadatos nwindex, pero el aumento del ajuste reduce la velocidad a la que se pueden ubicar los paquetes individuales o los elementos de metadatos. Este ajuste se aplica de inmediato.

La base de datos de sesión no tiene un ajuste de fidelidad, ya que no genera archivos de índice.

packet.integrity.flush,meta.integrity.flush, session.integrity.flush

Este ajuste controla si la base de datos impone una operación de sincronización en el sistema de archivos cuando termina de escribir en un archivo. El valor predeterminado es `sync` , lo cual significa que cuando se cierra un archivo, habrá un retraso significativo mientras los datos se escriben en almacenamiento no volátil. Puede ser necesario configurar este valor en `normal` para lograr mayores tasas de escritura sostenida, especialmente en un Decoder. Este ajuste se aplica en el siguiente archivo creado. Por lo tanto, se espera que se produzca al menos una sincronización más si el valor recién se cambió a `normal` .

Si se producen pérdidas de paquetes y packet.integrity.flush está configurado en `sync` , configúrelo en `normal` y monitoree. Mantenga los ajustes de vaciado de sesión y de metadatos en `sync` . Si las pérdidas de paquetes continúan causando problemas, configure las tres opciones en `normal` y monitoree.

packet.write.block.size,meta.write.block.size, session.write.block.size

El tamaño del bloque representa la cantidad de datos que se asigna a la vez dentro de cada archivo de base de datos. Los tamaños de bloque más grandes pueden proporcionar mayores tasas de rendimiento y de compresión, y pueden mejorar la tasa a la cual los artículos se pueden recuperar de la base de datos de forma secuencial. Sin embargo, los tamaños de bloques más grandes tienen un impacto perjudicial en la velocidad de lectura aleatoria de los elementos de paquetes y metadatos comprimidos. Este ajuste se aplica de inmediato.

packet.compression,meta.compression

Estos parámetros controlan si las bases de datos comprimen los datos. La compresión reduce la cantidad de almacenamiento que necesita cada base de datos, pero puede tener un impacto perjudicial considerable en la velocidad a la cual los elementos se escriben en la base de datos y se recuperan de ella. Los cambios se aplican de inmediato cuando se crea el siguiente archivo.

A partir de NetWitness Suite 10.4, los valores válidos para este parámetro son `gzip` , `bzip2` , `lzma` o `none` . `gzip` es el algoritmo recomendado cuando se usa compresión, ya que proporciona un buen equilibrio entre rendimiento y ahorro de espacio. `bzip2` y `lzma` pueden lograr un mejor ahorro de espacio, pero la desventaja en la velocidad es sustancial y es probable que solo se deba considerar para bajas velocidades de recopilación y cuando el espacio de almacenamiento es escaso.

packet.compression.level,meta.compression.level

Puede utilizar estos ajustes para mejorar la forma en que se comportan los algoritmos de compresión. No se aplican cuando la compresión está deshabilitada. Los valores válidos son entre 0 y 9. El valor predeterminado de cero implica permitir que el software elija el mejor ajuste para velocidad y compresión. Los valores entre 1 y 9 se usan como una escala variable entre rendimiento 1) y compresión (9). Generalmente, el valor 9 da la mejor compresión para un algoritmo determinado, pero el peor rendimiento. Generalmente el mejor ajuste se encuentra en algún lugar en el medio, que es cero.

hash.algorithm

Este ajuste controla la forma en que se aplica hash a los archivos de la base de datos. El valor predeterminado es `none` , de modo que no se aplica hash. Los valores válidos son `none` , `sha256` , `sha1` o `md5` . Se puede aplicar hash a los archivos de la base de datos para proporcionar pruebas de que no se han manipulado desde que se cerraron. La aplicación de hash requiere mucho tiempo y afecta el rendimiento de la recopilación cuando está activada. Este cambio se aplica de inmediato.

hash.databases

Este ajuste controla a cuáles bases de datos se aplica hash. Los valores válidos son `session` , `meta` y `packet` , y se separan con coma cuando se aplica hash a varias bases de datos. Este cambio se aplica de inmediato.

hash.dir

Normalmente este ajuste está vacío, lo que significa que el archivo de hash se crea en el mismo directorio que el archivo de base de datos con hash. Si se define este ajuste, el archivo hash se escribe en el directorio especificado en su lugar. Esto podría ser alguna forma de almacenamiento de una sola escritura para resistencia contra la manipulación de hash.

Los archivos hash son pequeños archivos XML que contienen el hash con codificación hexadecimal junto con metadatos sobre el archivo de la base de datos al que se aplicó hash.

Nodos de configuración del índice

En este tema se describen los nodos de configuración del índice. Los siguientes nodos de configuración de índice son algunos de los elementos avanzados de configuración de la base de datos de NetWitness Core que no cambian con frecuencia.

index.dir

La configuración `index.dir` controla dónde se almacenan los archivos que utiliza el índice. Esta configuración es compatible con la misma sintaxis que las configuraciones `packet.dir`, `meta.dir` y `session.dir`.

index.dir.warm

El almacenamiento de nivel semiactivo para segmentos del índice. Esta configuración es compatible con la misma sintaxis que `packet.dir.warm`, `meta.dir.warm` y `ession.dir.warm`.

index.dir.cold

El almacenamiento de nivel inactivo para segmentos del índice. Esta configuración es compatible con la misma sintaxis que `packet.dir.cold`, `meta.dir.cold` y `session.dir.cold`.

index.slices.open

Este ajuste controla la cantidad de segmentos del índice que el índice mantiene abiertos. Los segmentos del índice se abren automáticamente según lo requieren las consultas. Cuando las consultas se completan, el motor del índice puede mantener abiertos los segmentos de modo que las consultas subsiguientes se ejecuten con mayor rapidez. Los segmentos creados de manera más reciente son los que se mantendrán abiertos, puesto que es más probable que los usen las consultas.

Si las consultas al índice requieren que este abra segmentos, estas se ejecutarán con mayor lentitud que si los segmentos ya estuvieran abiertos. Por lo tanto, este parámetro se debe ajustar de modo que la mayoría de las consultas ejecutadas contra el índice funcionen en segmentos abiertos. Sin embargo, cada segmento del índice abierto consume algunos recursos, como identificadores de archivos y memoria. Si hay demasiados segmentos del índice abiertos, el rendimiento total del servicio puede verse afectado.

Debe configurar este parámetro de modo que los segmentos del índice abiertos cubran la mayor parte de los rangos de tiempo que necesitará la mayoría de las consultas. Por ejemplo, si la mayoría de las consultas se aplica a las últimas dos semanas y se crean segmentos del índice cada ocho horas, hay 14 días x tres segmentos diarios, lo cual significa que se crearon 42 segmentos en las últimas dos semanas. Por lo tanto, index.slices.open se podría configurar en 42 de modo que solo se mantengan abiertos los segmentos que probablemente se usarán.

Si este parámetro se configura en 0, se mantienen abiertos todos los segmentos hasta el siguiente guardado del índice. En este escenario, lo único que limita la cantidad de segmentos abiertos en el proceso es la cantidad de segmentos del índice.

page.compression

Obsoleto. Las versiones del índice de NetWitness Core entre 9.8 y 10.2 eran compatibles con dos algoritmos de compresión de índice distintos y puede escoger entre ellos mediante esta configuración. A partir de la versión 10.3, el único valor recomendado es el valor predeterminado `huffhybrid` .

save.session.count

Esta configuración controla con qué frecuencia se guarda automáticamente el índice cuando se insertan nuevas sesiones. Si el valor de save.session.count es mayor que 0, cada vez que se agregan más de `save.session.count` sesiones al índice, este se guarda automáticamente. Si `save.session.count` se configura en 0, esta función se deshabilita y el índice no se guarda automáticamente cuando se agregan nuevas sesiones.

`save.session.count` se puede usar para implementar un patrón de guardado automático que se basa en el volumen de datos que ingresan al índice. Esto es útil porque permite que un sistema cargado ligeramente genere puntos de guardado con menor frecuencia.

Para obtener más información sobre el tema de los guardados del índice, consulte la sección de esta guía relacionada con las Técnicas de optimización .

reindex.enable

Esta configuración controla la operación del reindexador en segundo plano .

Nodos de configuración de SDK

En este tema se describen los nodos de configuración de SDK que afectan a la base de datos. En cada servicio Core hay algunos elementos de configuración adicionales que afectan la base de datos, pero que no afectan realmente la manera en que la base de datos almacena o recupera los datos. Estas configuraciones existen en la carpeta `/sdk/config` .

max.concurrent.queries

Esta configuración controla cuántas operaciones de consulta se permiten simultáneamente en la base de datos. La autorización de más operaciones de consulta simultáneas puede mejorar la capacidad de respuesta general para más usuarios, pero si la carga de consultas del servicio Core está muy enlazada a las I/O, un valor max.concurrent.queries alto puede tener un efecto perjudicial. Se recomienda un valor cercano a la cantidad de cores del sistema, incluido el Hyper Threading. De esta forma, en un dispositivo con 16 cores, el valor debe estar en torno a 32. Reste algunos hilos de ejecución de agregación e hilos de ejecución de respuesta generales del sistema. Reste algunos más si se trata de un sistema híbrido (por ejemplo, un Decoder y un Concentrator que se ejecutan en el mismo dispositivo). No hay un número mágico, pero un valor entre 16 y 32 debe funcionar bien.

max.pending.queries

Esta configuración controla el tamaño de trabajos pendientes para el motor de consultas de la base de datos. Los valores más altos permiten que la base de datos ponga en línea de espera más operaciones para ejecución. Una consulta en línea de espera no avanza hacia su ejecución. Por lo tanto, puede ser más útil hacer que el sistema produzca errores cuando la línea de espera está llena que permitir que la línea de espera crezca mucho. Sin embargo, en un sistema que ejecuta principalmente operaciones por lotes, como informes, puede que no sea perjudicial tener una línea de espera grande.

cache.window.minutes

Esta configuración controla una función del motor de consultas que tiene como objetivo mejorar la capacidad de respuesta de las consultas cuando hay una gran cantidad de usuarios simultáneos. Para obtener más información sobre la ventana de caché, consulte Técnicas de optimización .

max.where.clause.cache

La caché de la cláusula Where controla cuánta memoria pueden consumir las operaciones de consulta que deben producir un conjunto de datos temporales grande para evaluar la clasificación y el conteo. Si se desborda el tamaño de la caché de la cláusula Where, la consulta funciona de todos modos, pero es mucho más lenta. Si la caché de la cláusula Where es demasiado grande, es posible que se asigne tanta memoria para las consultas que el servicio se vería forzado a un intercambio o quedaría sin memoria. Por lo tanto, este valor multiplicado por max.concurrent.queries siempre debe ser mucho menor que el tamaño de la RAM física. Esta configuración comprende los tamaños en la forma de un número seguido de una unidad, por ejemplo `1.5 GB` .

max.unique.values

Los valores únicos máximos limitan la cantidad de memoria que puede consumir la función Valores de SDK. Valores de SDK produce una lista ordenada de valores únicos. A fin de generar resultados precisos, puede que necesite combinar grandes cantidades de valores únicos de muchos segmentos. Este conjunto de valores combinados se debe mantener en la memoria y este parámetro existe para establecer un límite en la cantidad de memoria que puede consumir. El valor predeterminado limitará el uso de la memoria a aproximadamente una décima parte de la RAM total.

query.level.1.minutes , query.level.2.minutes , query.level.3.minutes

Esta configuración está disponible en NetWitness Suite 10.4 y en versiones anteriores.

En NetWitness Suite 10.4 y anteriores, la base de datos principal admite tres niveles de prioridad de consulta. A cada usuario se asigna uno de los niveles de prioridad. Por lo tanto, se pueden definir hasta tres grupos de usuarios con fines de ajuste del rendimiento. Estas configuraciones controlan cuánto tiempo se permite ejecutar consultas a cada nivel de usuario. Por ejemplo, los usuarios con privilegios inferiores pueden tener un valor menor de modo que no puedan usar todos los recursos del servicio Core con consultas de ejecución prolongada.

query.timeout

Esta configuración está disponible en NetWitness Suite 10.5 y en versiones superiores.

Los niveles de consulta se reemplazaron en NetWitness Suite 10.5 y superior por tiempos de espera de consulta por cuenta de usuario. Para las conexiones de confianza, estos tiempos de espera se configuran en el servidor de NetWitness Suite. Para cuentas en servicios principales, hay un nuevo nodo de configuración bajo cada cuenta, denominado `query.timeout` , que corresponde a la cantidad máxima de tiempo, en minutos, que se puede ejecutar cada consulta. Si este valor se configura en cero, significa que el servicio principal no impondrá ningún tiempo de espera de consulta.

max.where.clause.sessions

Esta configuración está disponible en NetWitness Suite 10.5 y en versiones superiores.

Impone un límite en la cantidad de sesiones que puede escanear una única consulta. Por ejemplo, si un usuario selecciona todos los metadatos de la base de datos, esta deja de procesar los resultados una vez que la cantidad de sesiones leídas para la consulta alcanza este valor de configuración. Un valor de 0 inhabilita este límite.

La cantidad de sesiones que se requieren para procesar por completo una consulta es igual a la cantidad de sesiones que coinciden con la cláusula WHERE de la consulta, bajo el supuesto de que todos los términos de la cláusula where tienen un índice apto. Si hay términos de la cláusula where que no están indexados, la base de datos debe leer más sesiones y metadatos, y alcanza este límite antes.

max.query.groups

Esta configuración está disponible en NetWitness Suite 10.5 y en versiones superiores.

Impone un límite en la cantidad de grupos únicos que se recopilan en una única consulta. Por ejemplo, si una consulta tiene una cláusula group by con múltiples metadatos que tienen altos conteos de valores únicos, la cantidad de memoria necesaria para esa consulta podría sobrepasar fácilmente la cantidad de RAM disponible en el servidor. De este modo, este límite existe para impedir que se produzcan condiciones de falta de memoria.

Si se configura un valor de 0, se inhabilita este límite.

packet.read.throttle

Esta es una configuración solo de Decoder que afecta al acceso a la base de datos de paquete. Si packet.read.throttle se configura en un valor mayor de 0, el Decoder intenta regular las lecturas de paquetes cuando detecta un conflicto de paquetes en la base de datos de paquete. Los números mayores proporcionan más regulación. Los cambios se aplican de inmediato.

cache.dir,cache.size

Todos los servicios de NetWitness Suite Core mantienen una pequeña caché de archivos de contenido crudo extraído del dispositivo. Estos parámetros controlan la ubicación (cache.dir) y el tamaño (cache.size) de este caché.

parallel.values

Esta configuración está disponible en NetWitness Suite 10.5 y en versiones superiores.

Esta configuración permite que las operaciones de valores de SDK se ejecuten en paralelo. Si se configura en 0, se deshabilita la ejecución en paralelo. Si se configura en un valor mayor de 0, representa la cantidad de hilos de ejecución que se crean cuando se ejecuta cada operación de valores de SDK. El valor máximo es la cantidad de CPU lógicos disponibles cuando se inició el proceso.

La configuración de un valor más alto para parallel.values es útil cuando hay pocos usuarios simultáneos, puesto que permite la ejecución de investigaciones más complejas con mayor rapidez. Si hay muchos usuarios simultáneos, es mejor usar un valor bajo, ya que se ejecutarán simultáneamente muchas operaciones de valores de SDK independientes.

parallel.query

Esta configuración está disponible en NetWitness Suite 10.5 y en versiones superiores.

Esta configuración se asemeja al ajuste parallel.values en que el valor máximo es la cantidad de CPU lógicos. La configuración de parallel.query en un valor específico debe considerar la cantidad de usuarios simultáneos para maximizar la utilización del CPU sin que se superen constantemente los recursos disponibles.

La configuración de un valor más alto para parallel.query es útil cuando hay pocos usuarios y consultas simultáneos, puesto que permite la ejecución de consultas más complejas con mayor rapidez. Si hay muchos usuarios y consultas simultáneos, es mejor usar un valor bajo, ya que se ejecutarán simultáneamente muchas operaciones de consultas de SDK independientes.

La velocidad de lectura de la base de datos de metadatos limita las operaciones de consultas; por lo tanto, si parallel.query se configura en un valor mayor de 4, es poco probable obtener resultados mucho mejores que con el valor predeterminado de 0. El mejor número que se debe usar para parallel.query dependerá del tipo de almacenamiento conectado. Experimente con distintos valores de parallel.query para determinar los mejores resultados para el sistema de almacenamiento.

Nodos de configuración por usuario

En este tema se describen los nodos de configuración por usuario. Hay configuraciones que influyen en las acciones que se permite realizar a los usuarios en la base de datos. Estas configuraciones se almacenan en el árbol de configuración en `/users/accounts/<username>/config` , donde `<username>` es el nombre del usuario al cual se aplican.

query.prefix

Un prefijo de consulta aplica un filtro a cada operación de consulta que realiza el usuario. Para implementar esto, se toman los valores de `query.prefix` y se les añade la cláusula Where de cada consulta con el uso del operador && (y) lógico. Para obtener más información acerca de las cláusulas Where, consulte Consultas .

query.level

Esta configuración está disponible en NetWitness Suite 10.4 y en versiones anteriores.

La configuración query.level asigna el nivel de consulta que tienen los usuarios para cada consulta que ejecutan. Estas influyen en el hecho de que query.level.1.minutes, query.level.2.minutes o query.level.3.minutes limite sus consultas.

query.timeout

Esta configuración está disponible en NetWitness Suite 10.5 y en versiones superiores.

La configuración query.timeout asigna la cantidad máxima de tiempo, en minutos, que un usuario puede ejecutar cada consulta. Para las conexiones de confianza, estos tiempos de espera se configuran en el servidor de NetWitness Suite. Para las cuentas en servicios principales, estas configuraciones se almacenan en el árbol de configuración en `/users/accounts/<username>/config` , donde `<username>` es el nombre del usuario al cual se aplican. Cuando este valor se configura en cero, el servicio principal no impone el tiempo de espera de consulta.

session.threshold

La configuración session.threshold asigna un umbral de sesión máximo para el usuario. Si se configura, este valor de umbral se asigna a todas las llamadas values que ejecuta el usuario. En esta guía se incluye un análisis detallado de las llamadas values y los umbrales.

Programador

En este tema se presenta brevemente el programador y se explica cómo se programan los comandos. Todos los servicios de NetWitness Core incluyen un programador incorporado que se encuentra en `/sys/config/scheduler` . Para usarlo, debe agregar el comando que desea ejecutar periódicamente mediante uno de dos mensajes:

`/sys/config/scheduler addInter` : Agregue un comando para que se ejecute en el intervalo especificado (cada N horas, minutos o segundos)

o

`/sys/config/scheduler addMil` : Agregue un comando para que se ejecute a la hora del día especificada o incluso en días específicos de la semana

Ejemplo

Por ejemplo, considere un caso de uso en el cual se eliminan todos los datos de paquetes que tienen más de siete días. Como no puede configurar el ajuste de packet.dir para implementar datos en función de un intervalo de tiempo, debe programar el comando `/database timeRoll` de modo que se ejecute cada cierto tiempo. En este ejemplo, cree un timeRoll para que se ejecute cada 20 minutos:

```
addIter minutes=20 pathname=/database msg=timeRoll params="type=packet
days=7"
```

Este comando agrega una tarea programada (que persiste entre reinicios del servicio) para que se ejecute cada 20 minutos, en el nodo `/database` , y excluya por antigüedad todos los datos de paquetes de más de siete días. El parámetro `params` se usa para transmitir todos los parámetros al comando especificado (en este caso `timeRoll`). Observe cómo se usan comillas en todos los parámetros integrados (`type` y `days`) de modo que no se interpreten como parámetros que se deben transmitir al comando `addIter` exterior. Si los parámetros dentro de `params` necesitan comillas, debe usar una barra invertida como carácter de escape en las comillas internas. Puede reescribirlo con comillas integradas, lo cual no altera el comando de ninguna manera:

```
addIter minutes="20" pathname="/database" msg="timeRoll"
params="type=\"packet\" days=\"7\""
```

Este comando tiene el mismo efecto que el original, pero demuestra cómo se usan los caracteres de escape en las transmisiones de parámetros complejas. Otros comandos útiles del programador son:

`/sys/config/scheduler print` : Imprima todos los comandos programados (también puede verlos si realiza una acción `ls` en el nodo del programador).

`/sys/config/scheduler delSched` : Elimine un comando programado mediante la transmisión del identificador que se muestra en el comando `print` (o `ls`).

Esto es una breve introducción del programador. Para obtener más información sobre los parámetros de comandos, envíe el mensaje `help` al nodo del programador y transmita el nombre del comando a través del parámetro `msg` . Para obtener más información, consulte el tema "Vista Explorar de servicios" de la *Guía de introducción de hosts y servicios* .

Rollover

En este tema se describen los dos mecanismos de transferencia. La base de datos opera como una línea de espera de tipo el primero en entrar, el primero en salir (FIFO). Los nuevos datos siempre se anexan a la base de datos y los datos más antiguos se eliminan automáticamente, si es necesario. Los datos en el centro de la base de datos son inmutables, lo que significa que no se pueden modificar.

Existen dos mecanismos para transferencia: síncrona y asíncrona.

Transferencia síncrona

Transferencia síncrona hace referencia a la configuración de transferencia que se aplica en respuesta a una operación de escritura en la base de datos. Esto significa que los datos se eliminan de la base de datos en respuesta directa a la necesidad de escritura de nuevos datos. La transferencia síncrona se configura mediante el establecimiento de los valores de tamaño en la configuración de `packet.dir` , `meta.dir` , `session.dir` y `index.dir` .

Se puede producir la transferencia síncrona en bases de datos de paquetes, metadatos y sesiones dentro de una operación de escritura. La transferencia síncrona en el índice se produce cuando se guarda el índice.

Transferencia asíncrona

La transferencia asíncrona hace referencia a la eliminación de archivos de base de datos que se produce cuando se emite un comando de transferencia explícita a la base de datos. Con mayor frecuencia, este tipo de transferencia se programa para ejecutarse de manera periódica con el programador incorporado del servicio Core. El usuario también puede solicitarlo explícitamente.

El comando de transferencia asíncrona es el mensaje `sizeRoll` presente en los nodos `/index` y `/database` del árbol de configuración. El mensaje en el nodo `/database` realiza una transferencia por tamaño en bases de datos de paquetes, metadatos y sesiones únicamente, mientras que el mensaje en el nodo `/index` puede realizar una transferencia simultánea en el índice y en las bases de datos de paquetes, metadatos y sesiones.

La sintaxis de parámetros del comando `sizeRoll` es la siguiente:

```
size-roll-params    = {type-param, space}, (max-size-param | min-free-
param | max-percent-param), {max-size-warm-param, space}
```

```
type-param          = "type=", {type-flag} , { ",", type-flag } ;
type-flag           = "packet" | "meta" | "session" ;
max-size-param      = "maxSize=", number, {space}, unit ;
max-percent-param   = "maxPercent=", number, {space}, unit ;
min-free-param      = "minFree=", number, {space}, unit ;
max-size-warm-param = "maxSizeWarm=", number, {space}, unit ;
unit                = "t" | "TB" | "g" | "GB" | "m" | "MB" ;
number              = ? decimal number ? ;
percentage          = ? number between 0 and 100 ? ;
```

El parámetro `type` controla las bases de datos que se consideran para quitar los datos más antiguos en función del tamaño total o del espacio restante. Si no se especifica el tipo en `/index` sizeRoll, solo se considera el índice para las operaciones de transferencia.

El parámetro `maxSize` establece un tamaño máximo actual de la base de datos o del índice. Si la base de datos es mayor que este tamaño, los datos más antiguos se eliminan primero (o se mueven al nivel semiactivo o inactivo, según la configuración) hasta que el tamaño total sea menor que `maxSize` . La operación `sizeRoll` determina qué datos son los más antiguos en todas las bases de datos y en el índice de acuerdo con los ID de sesión. Las entradas de sesiones o índice con los ID de sesión más bajos se eliminan primero. Posiblemente se incluye le eliminación de bases de datos de metadatos y paquetes huérfanas por la eliminación de entradas de la base de datos de sesiones. Los datos del índice se eliminan si se quitan las sesiones a las que hace referencia.

El parámetro `maxSizeWarm` establece un tamaño máximo actual en el nivel *semiactivo* , pero por lo demás se comporta de forma idéntica al parámetro `maxSize` . Cuando los datos se eliminan en el nivel semiactivo, se mueven al nivel inactivo (si está configurado) o se eliminan.

El parámetro `maxPercent` establece un porcentaje máximo de todos los volúmenes de todas las bases de datos transmitidas en el parámetro `type` combinadas. Cuando se excede, los datos más antiguos se eliminan primero hasta que el tamaño total es inferior a maxPercent para los volúmenes totales.

El parámetro `minFree` establece un espacio libre mínimo permitido en los volúmenes antes de que se eliminen los datos más antiguos.

Cada llamada a la operación sizeRoll proporciona un solo paso por la base de datos para eliminar archivos. Cuando se completa la operación, la utilización del tamaño actual de la base de datos habrá cumplido con los criterios que especifican los parámetros `maxSize` , `maxPercent` o `minFree` y el parámetro `maxSizeWarm` opcional. Por lo tanto, esta operación se puede programar periódicamente para asegurarse de que la base de datos pueda seguir funcionando de manera ininterrumpida.

Ejemplo

El siguiente ejemplo muestra una típica entrada de programador de sizeRoll para un Archiver:

```
pathname=/index minutes=5 msg=sizeRoll params="type=meta,session,packet
maxSize=25TB maxSizeWarm=150TB"
```

Esta entrada de programador especifica que cada cinco minutos, la base de datos se asegura de que el tamaño máximo de los metadatos, las sesiones, los paquetes y el índice no exceda los 25 terabytes en el nivel activo y los 150 terabytes en el nivel semiactivo.

Consultas

En este tema se describe la sintaxis de las consultas en la base de datos. Hay tres mecanismos principales para realizar consultas en la base de datos, las llamadas `query` , `values` y `msearch` en la carpeta `/sdk` de cada servicio principal.

La llamada `query` devuelve elementos de metadatos de la base de datos de metadatos, posiblemente mediante el uso del índice para una recuperación rápida.

La llamada `values` devuelve grupos de valores de metadatos únicos ordenados según algunos criterios. Está optimizada para devolver un subconjunto de los valores únicos ordenados según una función de agregado, como conteo.

La llamada `msearch` toma como su entrada los términos de búsqueda de texto y devuelve las sesiones que coinciden con los términos de búsqueda. Puede buscar dentro de los índices, los metadatos, los paquetes crudos o los registros crudos.

`query` Sintaxis

El mensaje `query` tiene la siguiente sintaxis:

```
query-params     = size-param, space, query-param, {space, start-meta-param}, {space, end-meta-param};
size-param       = "size=", ? integer between 0 and 1,677,721 ? ;
query-param      = "query=", query-string ;
start-meta-param = "id1=", metaid ;
end-meta-param   = "id2=", metaid ;
metaid           = ? any meta ID from the meta database ? ;
```

Los parámetros `id1` , `id2` y `size` forman un mecanismo de paginación para el retorno de una gran cantidad de resultados de la base de datos. Su uso beneficia principalmente a los desarrolladores que escriben aplicaciones directamente contra la base de datos de NetWitness Core. Normalmente, los resultados se devuelven en el orden de los datos más antiguos a los más recientes (los ID de metadatos más altos son siempre más recientes). Para devolver los resultados del más reciente al más antiguo, invierta los ID de tal manera que `id1` sea mayor que `id2` . Esto tiene una leve pérdida de rendimiento, ya que la cláusula where debe estar completamente evaluada antes de que el procesamiento en orden inverso pueda comenzar.

Cuando el tamaño se deja fuera o se establece en cero, el sistema retrocede todos los resultados sin paginación. En el caso de la interfaz RESTful, esto da lugar a que se devuelva la respuesta completa con codificación fragmentada. El protocolo nativo devuelve los resultados a través de varios mensajes.

El parámetro `query` es una cadena de comandos `query` con su propia sintaxis específica de NetWitness:

```
query-string      = select-clause {, where-clause} {, group-by-clause {,
order-by-clause } } ;
select-clause     = "select ",  ( "*" | meta-or-aggregate {, meta-or-
aggregate} ) ;
where-clause      = " where ",  { where-criteria } ;
meta-or-aggregate = meta | aggregate_func, "(", meta-key, ")" ;
aggregate_func    = "sum" | "count" | "min" | "max" | "avg" | "distinct"
| "first" | "last" | "len" | "countdistinct" ;
group-by-clause   = " group by ", meta-key-list
meta-key-list     = meta-key {, meta-key-list}
order-by-clause   = " order by ", order-by-column
order-by-column   = meta-or-aggregate { "asc" | "desc" } {, order-by-
column}
```

La cláusula `select` permite especificar `*` para devolver todos los metadatos en todas las sesiones que coincidan con la cláusula where, o un conjunto de nombres de campos de metadatos y funciones de agregado para seleccionar un subconjunto de los metadatos con cada sesión.

La cláusula `select` puede contener nombres de clave de metadatos cuyo nombre cambió. Los campos que aparecen en el conjunto de resultados como consecuencia del cambio de nombre de una clave en la cláusula `select` se devolverán con el nombre de la clave de metadatos que coincide con el nombre utilizado en la cláusula `select` . Por ejemplo, si la clave `port_src` se utiliza para cambiar el nombre de `tcp.srcport` , una consulta que contiene `select port_src` solo devolverá campos `port_src` , incluso si los metadatos subyacentes tuvieran el tipo `tcp.srcport` .

Las funciones de agregación tienen el siguiente efecto en el conjunto de resultados de las consultas.

Función	Resultado
`sum`	Agrega todos los valores de metadatos juntos; solo funciona con números
`count`	El total de campos de metadatos que se han devuelto
`min`	El valor mínimo que se ve
`max`	El valor máximo que se ve
`avg`	El valor promedio para el número
`distinct`	Devuelve una lista de todos los valores únicos que se ve

`countdistinct`	Devuelve la cantidad de valores únicos obtenidos. `countdistinct` es equivalente a la cantidad de metadatos que habría devuelto la función distinct.
`first`	Devuelve el primer valor que se ve
`last`	Devuelve el último valor que se ve
`len`	Convierte todos los valores de campo a una longitud UInt32 en lugar de devolver el valor real. Esta longitud es el número de bytes para almacenar el valor real, no la longitud de la estructura almacenada en la base de datos de metadatos. Por ejemplo, la palabra “NetWitness” devuelve una longitud de 10. Todos los campos IPv4, como `ip.src` , devuelven 4 bytes.

`where` Cláusulas

La cláusula `where` es una especificación de filtro que permite seleccionar sesiones de la recopilación mediante el índice.

Sintaxis:

```
where-criteria    = criteria-or-group, { space, logical-op, space,
criteria-or-group } ;
criteria-or-group = criteria | group ;
criteria          = meta-key, ( unary-op | binary-op meta-value-ranges )
;
group             = ["~"], "(" where-clause ")" ;
logical-op        = "&&" | "||" ;
unary-op          = "exists" | "!exists" ;
binary-op         = "=" | "!=" | "<" | ">" | ">=" | "<=" | "begins" |
"contains" | "ends" | "regex" ;
meta-value-ranges = meta-value-range, { ",", meta-value-range } ;
meta-value-range  = (meta-value | "l" ), [ "-", ( meta-value | "u" ) ] ;
meta-value        = number | quoted-value | ip-address | mac-address |
relative-time ;
quoted-value      = ( '"' text '"' ) | ( '"' date-time '"' ) ;
relative-time     = "rtp(" , time-boundary , "," ,  positive-integer ,
time-unit, ")" ;
time-boundary     = "earliest" | "latest" ;
```

```
positive-integer  = ? any non-negative integral number ?
time-unit         = "s" | "m" | "h" ;
```

Cuando se especifican criterios de regla, se espera que la parte `meta-value` de la cláusula coincida con el tipo de metadatos que especifica `meta-key` . Por ejemplo, si la clave es `ip.src` , `meta-value` debe ser una dirección IPv4.

Las consultas que usan un nombre `meta-key` coincidirán con elementos de metadatos correspondientes al nombre `meta-key` y también a los nombres de cualquier "cambio de nombre" especificado para la clave. Consulte "Cambio de nombre de clave" en el tema Personalización del índice para obtener detalles acerca del cambio de nombre de clave.

Operadores de consulta

En la tabla siguiente se describe la función de cada operador.

Operador	Función
=	Coincide con sesiones que contienen exactamente el mismo valor de metadatos. Si se especifica un rango de valores, cualquiera de los valores se considera una coincidencia.
!=	Coincide con todas las sesiones que no coincidirían con la misma cláusula como si estuvieran escritas con el operador = .
<	En el caso de valores numéricos, coincide con sesiones que contienen metadatos con el valor numérico menor que el lado derecho. Si el lado derecho es un rango, se considera el primer valor del rango. Si se especifican varios rangos, el comportamiento es indefinido. Para metadatos de texto, se realiza una comparación lexicográfica.
<=	El mismo comportamiento que < , pero las sesiones que contienen metadatos que igualan exactamente el valor también se consideran coincidencias.
>	Similar al operador < , pero coincide con sesiones en las cuales el valor numérico es mayor que el lado derecho. Si el lado derecho es un rango, se considera el último valor del rango para la comparación.
>=	El mismo comportamiento que > , pero las sesiones que contienen metadatos que igualan exactamente el valor también se consideran coincidencias.
`begins`	Coincide con sesiones que contienen valor de metadatos de texto que comienzan con los mismos caracteres que el lado derecho.

`ends`	Coincide con sesiones que contienen metadatos de texto que terminan con los mismos caracteres que el lado derecho.
`contains`	Coincide con sesiones que contienen metadatos de texto que incluyen la subcadena dada en el lado derecho.
`regex`	Coincide con sesiones que contienen metadatos de texto que coinciden con la expresión regular dada en el lado derecho. El análisis de la expresión regular se maneja mediante boost::regex.
`exists`	Coincide con sesiones que contienen cualquier valor de metadatos con la clave de metadatos dada.
`!exists`	Coincide con sesiones que no contienen valor de metadatos con la clave de metadatos dada.
`length`	Coincide con sesiones que contienen valores de metadatos de texto de cierta longitud. La expresión en el lado derecho debe ser un número no negativo.

Valores de texto

El sistema espera valores de texto entre comillas. A menos que se pueda analizar como tiempo (consulte a continuación), un valor entre comillas se interpreta como texto.

Direcciones IP

Las direcciones IP se pueden expresar mediante representaciones de texto estándar para las direcciones IPv4 e IPv6. Además, la consulta puede usar la notación CIDR para expresar un rango de direcciones. Si se usa la notación CIDR, se expande al rango de valores equivalente.

Direcciones MAC

Una dirección MAC se puede especificar mediante una notación de dirección MAC estándar: `aa:bb:cc:dd:ee:ff`

Expresiones de fecha y hora

En NetWitness Suite, las fechas se representan mediante tiempo Unix, que es la cantidad de segundos desde el 1 de enero de 1970 UTC. En las consultas, se puede expresar el tiempo como esta cantidad de segundos, o puede usar la representación de cadena. La representación de cadena para la fecha y la hora es `"YYYY-mmm-DD HH:MM:SS"` . Una abreviatura de tres letras representa el mes. También puede expresar el mes como un número de dos dígitos, 01–12.

Los valores de hora deben ir entre comillas.

Se espera que todas las horas que se expresan en las consultas estén en UTC.

Puntos de tiempo relativos

Los puntos de tiempo relativos permiten que una cláusula where haga referencia a un valor en algún desplazamiento fijo relativo a los metadatos de tiempo más tempranos o más recientes observados en la recopilación.

Una expresión de punto de tiempo relativo tiene la sintaxis `rtp(boundary, duration)` .

El límite es `earliest` o `latest` .

La duración es una expresión de horas, minutos o segundos. Por ejemplo, `24h` , `60m` o `60s` .

Los puntos de tiempo relativos solo se pueden utilizar en operaciones de SDK, donde hay una recopilación desde la cual se obtienen los límites para los metadatos de tiempo más tempranos y más recientes.

Los puntos de tiempo relativos solo funcionan en tipos de metadatos indexados. Los tipos de metadatos indexados predeterminados son `time` y `event.time` .

Ejemplos:

```
Last 90m of collection time:
time = rtp(latest, 90m) - u


First 2 days of event time:
event.time = l - rtp(earliest, 48h)
```

Valores de rango especiales

Generalmente, los rangos se expresan con la sintaxis * `smallest` * - * `largest` *, pero puede usar algunos valores especiales de marcador de posición en las expresiones de rango. Puede usar la letra `l` para representar el límite inferior de todos los valores de metadatos como el inicio del rango, y `u` para representar el límite superior. Los límites se determinan con la observación del valor de metadatos menor o mayor que se encuentra en el índice de todos los valores de metadatos que ya ingresaron en el índice.

Si usa la etiqueta `l` o `u` , no debe estar entre comillas.

Por ejemplo, la expresión `time = "2014-may-20 11:57:00" - u` coincidiría con todas las horas desde el 20 de mayo de 2014 a las 11:57:00 h hasta la hora más reciente que se encuentra en la recopilación.

Observe que es fácil confundir una expresión de rango con una cadena de caracteres. Asegúrese de que los valores de texto que contienen − estén entre comillas y que los guiones dentro de las expresiones de rango no estén dentro de texto entre comillas.

Cláusula `group by` (desde 10.5)

La API de consulta tiene la capacidad de generar grupos agregados a partir de los resultados de una llamada query. Esto se hace mediante una cláusula `group by` en la consulta. Cuando se especifica `group by` , el conjunto de resultados para la consulta se subdivide en grupos. Cada grupo de resultados se identifica de manera única con los valores de metadatos que se indican en la cláusula group by.

Por ejemplo, considere la consulta `select count(ip.dst)` . Esta consulta devuelve un conteo de todos los metadatos ip.dst en la base de datos. Sin embargo, si agrega una cláusula `group by` como esta: `select count(ip.dst) group by ip.src` , la consulta devuelve un conteo de los metadatos ip.dst que se encuentran para cada ip.src único.

A partir de NetWitness Suite versión 10.5, puede utilizar hasta 6 campos de metadatos en una cláusula `group by` .

La cláusula `group by` tiene en parte la misma funcionalidad que la llamada `values` , pero ofrece una cantidad considerablemente mayor de grupos avanzados a expensas de tiempos de consulta más prolongados. La producción de los resultados de una consulta agrupada implica leer los metadatos de la base de datos de metadatos de todas las sesiones que coinciden con la cláusula `where` , mientras que una llamada values puede producir sus agregados con tan solo leer el índice.

La cláusula `select` define el contenido de cada grupo que devuelve la consulta. La cláusula `select` puede contener cualquiera las funciones de agregado o de los campos de metadatos seleccionados. Si se seleccionan múltiples agregados, el resultado de la función de agregado se define para cada grupo. Si se seleccionan campos no agregados, los campos de metadatos se devuelven en lotes para cada grupo.

El conjunto de resultados de una consulta `group by` se codifica con las siguientes reglas:

1. Todos los elementos de metadatos asociados a un grupo se entregan con el mismo número de grupo.
2. Los primeros elementos de metadatos que se devuelven al grupo identifican la clave del grupo. Por ejemplo, si la cláusula `group by` especifica `group by ip.src` , el primer elemento de metadatos de cada grupo será `ip.src` .
3. Los elementos de metadatos no agregados normales se devuelven después de `group key` , pero todos tendrán el mismo número de grupo que los metadatos de clave del grupo.
4. A continuación se devuelven los campos de metadatos de resultados agregados para cada grupo.
5. Todos los campos dentro de un grupo se devuelven juntos. Los diversos resultados de grupos no se entrelazarán.

Si falta uno de los elementos de metadatos `group by` en una de las sesiones que coincidieron con la cláusula `where` , ese campo de metadatos se trata como NULO en lo que respecta a ese grupo. Cuando se devuelven los resultados para ese grupo, las partes con valor NULO de la clave del grupo se omitirán de los resultados del grupo, puesto que la base de datos no tiene el concepto de NULO.

La semántica de una consulta `group by` difiere de una base de datos como SQL en términos de los campos de metadatos que se devuelven. En las bases de datos SQL, debe seleccionar explícitamente las columnas `group by` en la cláusula `select` si desea que se devuelvan en el conjunto de resultados. La base de datos de NetWitness Core devuelve siempre implícitamente las columnas de grupo en primer lugar.

Una consulta con una cláusula `group by` respeta el parámetro `size` del conjunto de resultados, si se proporciona uno. Sin embargo, debido a la naturaleza de la agrupación, impone una carga adicional en el llamador para paginar y reformar grupos en caso de que se solicite un conjunto de resultados de tamaño fijo. Por esta razón, no debe especificar un tamaño de resultado explícito cuando se hace una llamada `group by` . Cuando no se especifica un tamaño explícito, el conjunto de resultados completo se entrega como resultados parciales.

En la siguiente tabla se describen los parámetros de configuración que debe respetar la base de datos y que limitan el impacto en los I/O o en la memoria de una consulta group by.

Parámetro	Función
`/sdk/config/max.query.groups`	Este es el límite de la cantidad de grupos que se pueden mantener en la memoria para calcular agregados. Este parámetro permite limitar el uso total de la memoria de la consulta.
`/sdk/config/max.where.clause.sessions`	Este es el límite de la cantidad de sesiones desde la cláusula where que se pueden procesar en una consulta. Este parámetro permite configurar un límite de la cantidad de sesiones que se deben leer desde las bases de datos de sesiones y metadatos para resolver una consulta.

Cláusula `order by` (desde 10.5)

Una cláusula `order by` se puede agregar a una consulta que contiene una cláusula `group by` . La cláusula `order by` hace que el conjunto de resultados agrupados se devuelva de manera ordenada.

Una cláusula `order by` consta de un conjunto de elementos de clasificación en orden ascendente o descendente. El orden se puede aplicar a cualquiera de los campos de datos que se devolverá en el conjunto de resultados. Esto incluye los metadatos que especifica la cláusula `select` , los resultados de las funciones de agregado que especifica la cláusula `select` o los campos de metadatos de `group by` .

La cláusula `order by` puede ordenar muchas columnas. No hay un límite en la cantidad de columnas `order by` que se permiten en la consulta, pero existe un límite práctico que consiste en que cada una de las columnas `order by` debe hacer referencia a algo que devolvieron las cláusulas `select` o `group by` . La clasificación de múltiples columnas se impone de manera lexicográfica, lo cual significa que si dos grupos tienen valores iguales para la primera columna, se clasifican por las segundas columnas. Si son iguales en la segunda columna, se clasifican por la tercera y así sucesivamente para la cantidad de columnas `order by` que se proporcionan.

La base de datos de NetWitness Core es única en el sentido de que cada uno de los grupos de resultados que devuelve una consulta puede tener muchos valores para una selección. Por ejemplo, es posible seleccionar todos los elementos de metadatos que coinciden con un tipo de metadatos y organizarlos en grupos, y es posible usar la función `distinct()` para devolver grupos de valores de metadatos distintos. Si una cláusula `order by` hace referencia a uno de los campos del grupo que tiene múltiples valores, el orden de clasificación se aplica de la siguiente manera:

1. Dentro de cada grupo, los campos con múltiples valores coincidentes se ordenan por la cláusula de orden
2. Todos los grupos se clasifican mediante la comparación de la primera aparición del campo ordenado que se encuentra dentro de cada grupo

La cláusula `order by` solo está disponible en consultas que tienen una cláusula `group by` , puesto que se requieren grupos para organizar los campos de metadatos en los registros distintos. Si desea clasificar una consulta arbitraria como si no se aplicara agrupación, use `group by sessionid` . Esto garantiza que los resultados se devuelven en grupos de sesiones o eventos distintos.

Las cláusulas `group by` se devuelven naturalmente en orden de clave de grupo ascendente, pero se puede usar una cláusula `order by` para devolver grupos en otro orden.

Si una columna `order by` no especifica `asc` o `desc` , el orden predeterminado es ascendente.

Ejemplos:

```
select countdistinct(ip.dst) GROUP BY ip.src ORDER BY countdistinct
(ip.dst)
select countdistinct(ip.dst) GROUP BY ip.src ORDER BY countdistinct
(ip.dst) desc
select countdistinct(ip.dst),sum(size) GROUP BY ip.src ORDER BY sum
(size) desc, countdistinct(ip.dst)
select sum(size) GROUP BY ip.src, ip.dst ORDER BY ip.dst desc
select user.dst,time GROUP BY sessionid ORDER BY user.dst
select * GROUP BY sessionid ORDER BY time
```

Llamada `values`

El índice proporciona una función `values` de bajo nivel para acceder a los valores de metadatos únicos que se almacenaron en el índice. Esta función permite a los desarrolladores realizar operaciones más avanzadas en grupos de valores de metadatos únicos.

La sintaxis de los parámetros de la llamada `values` :

```
values-params         = field-name-param, space, where-param, space,
size-param, {space, flags-param} {space, start-meta-param}, {space, end-
meta-param}, {space, threshold-param}, {space, aggregate-func-param},
{space, aggregate-field-param}, {space, min-param}, {space, max-param} ;
field-name-param      = "fieldName=", meta-key ;
where-param           = "where=", where-clause ;
size-param            = "size=", ? integer between 1 and 1,677,721 ? ;
start-meta-param      = ? same as query message ?
end-meta-param        = ? same as query message ?
flags-param           = "flags=", {values-flag, {"," values-flag} } ;
values-flag           = "sessions" | "size" | "packets" | "sort-total" |
"sort-value" | "order-ascending" | "order-descending" ;
threshold-flag        = "threshold=", ? non-negative integer ? ;
aggregate-func-param  = "aggregateFunction=", { aggregate-func-flag } ;
aggregate-func-flag   = "count" | "sum" ;
aggregate-field-param = "aggregateFieldName=", meta-key ;
min-param             = "min=", meta-value ;
max-param             = "max=", meta-value ;
```

La llamada `values` proporciona la función de devolver un conjunto de valores de metadatos únicos para una clave de metadatos determinada. Para cada valor único, la llamada `values` puede proporcionar un conteo total de agregación. El parámetro de marcas controla la función que se utiliza para generar el total.

Parámetros

En la siguiente tabla se describe la función de cada parámetro.

Parámetro	Función
`fieldName`	Este es el nombre de la clave de metadatos para la cual se recuperan valores únicos. Por ejemplo, si `fieldName` es `ip.src` , esta función devuelve los valores de IP de origen únicos de la recopilación. Si `fieldName` se refiere a una clave con referencias de cambio de nombre, el resultado se define como el conjunto combinado de valores de campo para el nombre de clave de metadatos determinado más todas las claves de metadatos de las referencias.
`where`	Esta es una cláusula `where` que filtra el conjunto de sesiones para el cual se devuelven los valores únicos. Por ejemplo, si `fieldName` es `ip.src` y la cláusula `where` es `ip.src = 192.168.0.0/16` , solo se devuelven valores en el rango de `192.168.0.0` a `192.168.255.255` . Para obtener información sobre la sintaxis de la cláusula `where` , consulte *Cláusulas where* .
`size`	El tamaño del conjunto de valores únicos para devolver. Esta función está optimizada para devolver un pequeño subconjunto de los posibles valores únicos en la base de datos.
`id1` , `id2`	Estos parámetros opcionales limitan el alcance de la búsqueda de valores únicos a una región específica de la base de datos de metadatos y el índice. Establecer los parámetros `id1` e `id2` en un rango limitado de la base de datos de metadatos es muy importante para realizar búsquedas rápidamente en recopilaciones grandes.
`flags`	Los indicadores controlan cómo se clasifican y totalizan los valores. Los indicadores se describen en la siguiente sección

	Indicadores de valores.
`threshold`	La configuración del parámetro `threshold` permite que la llamada `values` realice un acceso directo para la recopilación del total asociado a cada valor una vez que se alcanza el umbral. El usuario que llama, cuando proporciona un umbral, puede reducir la cantidad de elementos de índice y metadatos que se debe recuperar de la base de datos. Si el parámetro `threshold` se omite o se establece en 0, esta optimización no se utiliza.
`aggregateFunction`	Parámetro opcional que se usa para cambiar el comportamiento predeterminado de conteo de sesiones, paquetes o tamaños a conteo o resumen del campo numérico que define `aggregateFieldName` . Se deben especificar ambos parámetros cuando se define alguno. Transmita `sum` o `count` para especificar el comportamiento que se ejecutará.
`aggregateFieldName`	El campo de metadatos en el que se ejecuta `aggregateFunction` . Los parámetros `aggregateFunction` y `aggregateFieldName` se deben especificar cuando se configura la marca `aggregate` . Realizar una llamada `values` con el uso de una de las funciones de agregado puede ser significativamente más lento que una llamada `values` que recopila totales de sesiones, paquetes o tamaños. El motivo de esto es que cada sesión que coincide con la cláusula `where` debe recuperarse de la base de datos de metadatos. Este escaneo hace que una gran parte de la consulta esté en el límite de I/O de los volúmenes de base de datos de metadatos. El tiempo que tarda la ejecución de una llamada `values` agregada es linealmente proporcional a la cantidad de sesiones que coinciden con la cláusula `where` .
`min` , `max`	El valor mínimo y máximo que debería devolver la llamada. Estos parámetros se usan para iterar (o paginar) en una cantidad de valores extremadamente grande, generalmente más valores que aquellos que podrían devolver una sola llamada. Se usan

principalmente en conjunto con las marcas `sort-value`, `sort-ascending` , de tal modo que el valor más alto devuelto se use en una llamada posterior como el valor del parámetro `min` . Los valores son exclusivos. Si se especificó `min="rsa"` y `rsa` es un valor válido, no se devolverá `rsa` , sino que el siguiente valor más alto.

Marcas de `values`

El parámetro `flags` controla la forma en que funciona la llamada values. Hay tres grupos de marcas que corresponden a los diferentes modos de funcionamiento, como se muestra en la siguiente tabla.

Marca	Descripción
`sessions` , `size` , `packets`	La llamada `values` permite especificar una de estas marcas para determinar cómo se calcula el total de cada valor. Si la marca es `sessions` , la llamada `values` devuelve un conteo de las sesiones que contienen cada valor. Si la marca es `size` , la llamada `values` totaliza el tamaño de todas las sesiones que contienen cada valor único e informa el tamaño total de cada valor único. Si la marca es `packets` , la llamada values totaliza la cantidad de paquetes de todas las sesiones que contienen cada valor único e informa el total de cada valor único.
`sort-total` , `sort-value`	Estas marcas controlan cómo se clasifican los resultados. Si la marca es `sort-total` , el conjunto de resultados se clasifica por orden de los totales recopilados. Si la marca es `sort-value` , los resultados se devuelven por orden según el orden de clasificación de los valores.
`order-ascending` , `order-descending`	Estos indicadores controlan el orden de clasificación del conjunto de resultados. Por ejemplo, si se clasifica por total en orden descendente, en primer lugar se devuelven los valores con el total mayor.

Ejemplo de llamada `values`

La vista Navegación en NetWitness Suite usa ampliamente la llamada `values` . La vista predeterminada genera llamadas que tienen la siguiente apariencia:

```
/sdk/values id1=198564099173 id2=1542925695937 size=20
flags=sessions,sort-total,order-descending threshold=100000
fieldName=ip.src where="time=\"2014-May-20 13:12:00\"-\"2014-May-21
13:11:59\""
```

En este ejemplo, la vista Navegación solicita valores únicos para `ip.src` . Solicita valores únicos de `ip.src` en el rango de tiempo dado. Solicita el conteo de sesiones que coinciden con cada `ip.src` y los resultados son los 20 valores `ip.src` principales cuando se clasifican por el conteo total de sesiones en orden descendente. Además, la vista Navegación tiene un rango de ID de metadatos para proporcionar una pista de optimización al motor de consulta.

Llamada `msearch`

El índice proporciona una función `msearch` de bajo nivel para realizar búsquedas de texto de todos los tipos de metadatos. Este tipo de búsqueda no requiere que los usuarios definan sus consultas en términos de tipos de metadatos conocidos. En su lugar, busca coincidencias en todas las partes de la base de datos. `msearch` se usa en la búsqueda de texto de la vista Eventos. Consulte el tema “Filtrar y buscar resultados en la vista Eventos” de la *Guía de Investigation y Malware Analysis* para obtener detalles acerca de las formas de búsqueda aceptadas y ejemplos.

Parámetros de `msearch` :

```
msearch-params  = search-param, {space, where-param}, {space, limit-
param}, {space, flags-param};
search-param    = "search=", ? free-form search string ? ;where-param
 = "where=", ? optional where clause ? ;
limit-param     = "limit=", ? optional session scan limit ? ;flags
    = "flags=", {msearch-flag, {"," msearch-flag} };
msearch-flag    = "sp" | "sm" | "si" | "ci" | "regex" ;
```

El algoritmo `msearch` funciona de la siguiente manera:

1. Se identifica un conjunto de sesiones del índice mediante la búsqueda de la intersección de tres conjuntos:
   - (Conjunto 1) Todas las sesiones de la base de datos
   - (Conjunto 2) Sesiones que coinciden con el parámetro de la cláusula `where`
   - (Conjunto 3) Si se especifica la marca `si` , sesiones con valores indexados que coinciden con el parámetro de cadena de búsqueda.

2. Si la búsqueda especifica el parámetro `sm` , se leen todos los elementos de metadatos del conjunto de sesiones identificado en el paso 1 y se escanean para ver si coinciden con el parámetro de cadena de búsqueda. Los elementos de metadatos se leerán desde el servicio más cercano al punto donde se ejecutó la búsqueda. Por ejemplo, si la búsqueda se realiza en un Broker, los elementos de metadatos se pueden leer desde el Concentrator más cercano al Broker, pero si se realiza en un Archiver, se leerán desde el propio Archiver.
3. Si la búsqueda especifica el parámetro `sp` , se leen todas las entradas de paquetes o registros crudos del conjunto de sesiones identificado en el paso 1 y se escanean para ver si coinciden con el parámetro de cadena de búsqueda. Los paquetes se leerán desde el servicio más cercano al punto donde se ejecutó la búsqueda. Por ejemplo, si la búsqueda se realiza en un Concentrator, los datos de paquetes se leerán desde el Decoder, pero si se realiza en un Archiver, se leerán desde el propio Archiver.
4. Las coincidencias de los pasos 2 y 3 se devuelven a medida que se encuentran hasta el punto en el cual se alcanza el parámetro `limit` . Limit especifica la cantidad máxima de sesiones para las cuales se escanearán metadatos y datos de paquetes. Si no se especifica, se escanea el conjunto de sesiones completo determinado en el paso 1.

Marcas de `msearch`

Marca Descripción

`sp`	Escanea datos de paquetes crudos
`sm`	Escanea todos los metadatos
`si`	Realiza búsquedas de índice para todos los parámetros de búsqueda antes de escanear metadatos
`ci`	Lleva a cabo una búsqueda que no distingue mayúsculas de minúsculas. Los resultados devueltos conservan las mayúsculas y las minúsculas.
`regex`	Trata el parámetro de búsqueda como una expresión regular. Se puede especificar solo una expresión regular, pero esta puede ser arbitrariamente compleja.

Modo de búsqueda de índice de `msearch`

El uso del modo de búsqueda de índice, que se especifica mediante la marca `si` , hace que los resultados se devuelvan de manera mucho más rápida que con cualquier otro modo. La principal limitación de este modo es que solo devuelve coincidencias en términos de texto que coinciden con valores de metadatos indexados por valor.

- El parámetro `si` se debe combinar con la marca `sm` . El parámetro `si` implica que la búsqueda solo obtiene coincidencias con metadatos indexados.
- El parámetro `si` se puede utilizar con búsquedas regex; sin embargo, solo habrá coincidencias con valores indexados de texto. Las direcciones IP y los números no coincidirán con regex.

Consejos para `msearch`

- Utilice siempre la cláusula `where` para especificar un rango de tiempo para la búsqueda.
- Para buscar rangos de direcciones IP, debe especificarlas en la cláusula `where` .
- Utilice el parámetro `limit` cuando no use el modo de búsqueda de índice. Sin él, las bases de datos de metadatos y paquetes leerán una cantidad excesiva de datos.

Procedimientos almacenados

Las llamadas `query` y `values` proporcionan más funcionalidad de búsqueda de bajo nivel. Para los casos de uso más avanzados, existen procedimientos almacenados en el nivel de servidor.

Uso de comillas en la sintaxis de una consulta

El analizador de consultas no distingue si se usan comillas simples o dobles en una declaración de consulta. Un valor entre comillas simples o dobles se trata como metadatos de texto.

El analizador de consultas intenta encontrar sentido a lo que se agrega en la declaración. No es muy estricto en cuanto a lo que aceptará.

Por ejemplo:

`reference.id=4752`

Esta cláusula identifica las sesiones que tienen un valor de metadatos `reference.id` con un valor *numérico* de 4,752.

`reference.id='4752'` o `reference.id="4752"`

Esta cláusula identifica las sesiones que tienen un valor de metadatos `reference.id` con un valor de *cadena* de `4752` .

Sin embargo, el motor de consultas compara de manera implícita como iguales los números y las cadenas que parecen números cuando los valores son semánticamente iguales. Por lo tanto, funciona con cualquier sintaxis.

Sin embargo, para lograr el rendimiento más eficiente, siempre es recomendable crear consultas de modo que su sintaxis coincida con los tipos de datos que genera el analizador.

Por ejemplo, si el analizador está creando `reference.id` como un tipo de datos numérico (por ejemplo, `uint32` o `uint64`), use la sintaxis numérica.

Si está creando `reference.id` como un tipo de datos de texto, use la sintaxis de cadena.

Personalización del índice

En este tema se describe cómo usar el archivo de índice personalizado para personalizar el índice. Cada servicio NetWitness NextGen se instala con una configuración de índice predeterminada que tiene como objetivo satisfacer las necesidades del índice para la mayoría de los usuarios del producto. Sin embargo, es posible indexar nuevas claves de metadatos para usar el índice con contenido personalizado que generó metadatos personalizados.

Ubicaciones del archivo de configuración del índice

La personalización del índice se lleva a cabo mediante cambios en el archivo de índice personalizado. La ubicación de este archivo es **/etc/netwitness/ng/index-<servicename>-custom.xml** , donde **<servicename>** corresponde al nombre del producto que está personalizando. Por ejemplo, el archivo de índice personalizado de Concentrator es /etc/netwitness/ng/index-concentrator-custom.xml.

Los productos Concentrator también incluyen un archivo que describe la configuración predeterminada del índice: **/etc/netwitness/ng/index-concentrator.xml** . Este archivo es útil como plantilla para mostrar cómo está formateado el archivo de índice personalizado.

Si realiza personalizaciones al índice en el archivo de índice personalizado, estas anulan los conflictos con la configuración predeterminada del índice.

Puede hacer cambios en el archivo de índice personalizado mientras el servicio está en ejecución. Cuando el servicio recibe un comando de guardado del índice, los cambios en el archivo de índice personalizado se leen y se aplican al índice.

Los cambios en el índice solo se pueden aplicar a los nuevos datos entrantes. Los datos no se pueden volver a indexar retroactivamente con una nueva configuración personalizada del índice, excepto mediante una reconstrucción del índice .

Entradas de configuración del índice

El archivo de índice personalizado es un documento XML. El elemento raíz de este documento es el elemento `language` y se incluye un elemento por clave de metadatos para describir cada índice personalizado. Cada elemento de la configuración personalizada del índice tiene la siguiente apariencia:

```
<key name="did" description="Decoder Source" level="IndexValues"
format="Text" valueMax="100"  />
```

Definiciones de cada atributo en este elemento: Atributo | Descripción -|- name | Nombre de la clave de metadatos que se indexará description | Descripción en lenguaje natural del tipo de metadatos level | Tipo de índice que se creará para esta clave de metadatos valueMax | Valores únicos máximos que se almacenarán para esta clave por segmento format | Formato de los datos que contienen todos los elementos de metadatos con este nombre de clave de metadatos.

Las siguientes secciones analizan estos parámetros de manera más detallada.

Nombres de metadatos

El nombre de metadatos que usa el índice se refiere al nombre de la clave de metadatos presente en cada elemento de metadatos en la base de datos de metadatos. Los Decoders generan estos nombres de metadatos durante el análisis. Los analizadores pueden optar por generar metadatos con cualquier nombre de clave de metadatos. Por lo tanto, el índice personalizado permite elegir cuáles de los elementos de metadatos que genera el Decoder se indexan.

Los nombres de clave de metadatos pueden tener 16 caracteres de largo e incluir solo letras o el carácter ".".

Tipos de datos

Cuando el Decoder genera elementos de metadatos, asigna un tipo de datos. Cada analizador puede elegir el tipo de datos de los metadatos que genera. Sin embargo, hay tipos de datos recomendados y estándar para cada una de las claves de metadatos predeterminadas. Por ejemplo, ip.src e ip.dst se almacenan como el tipo de metadatos IPv4 y alias.host se almacena como el tipo de metadatos texto. Cada analizador debe acordar el formato de datos para cada clave de metadatos que genera el Decoder.

Cuando se agrega un índice personalizado a Concentrator, el tipo de datos del índice personalizado debe coincidir con el formato de los datos que genera el Decoder. Si los tipos no coinciden, Concentrator intenta conversiones de los metadatos generados al tipo especificado para el índice personalizado. Sin embargo, estas conversiones pueden fallar y el índice resultante puede producir resultados indeterminados.

Asimismo, cuando muchos Decoders y Concentrators trabajan en conjunto como parte de una instalación de NetWitness, deben acordar los tipos para cada clave de metadatos. Los conflictos de tipos de metadatos entre los servicios NetWitness NextGen pueden llevar a un comportamiento indeterminado.

En la siguiente tabla se muestran los tipos de metadatos compatibles con los servicios NetWitness NextGen.

Tipo	Tamaño en bytes	Descripción
Int8	1	Entero de 8 bits con signo
UInt8	1	Entero de 8 bits sin signo
Int16	2	Entero de 16 bits con signo

UInt16	2	Entero de 16 bits sin signo
Int32	4	Entero de 32 bits con signo
UInt32	4	Entero de 32 bits sin signo
Int64	8	Entero de 64 bits con signo
UInt64	8	Entero de 64 bits sin signo
UInt128	16	Entero de 128 bits sin signo
Float32	4	Número de punto flotante de 32 bits, precisión sencilla
Float64	8	Número de punto flotante de 64 bits, doble precisión
TimeT	8	Registro de fecha y hora epoc de Unix
Binaria	1-255	Datos binarios arbitrarios
Texto	1-255	Datos de texto con codificación UTF-8
IPv4	4	Bytes de dirección IPv4
IPv6	16	Bytes de dirección IPv6
MAC	6	Bytes de dirección MAC

Cuando se define un índice personalizado, es importante usar el mejor tipo de datos para los metadatos. Por ejemplo, nunca almacene direcciones IP como texto, puesto que la representación de texto requiere más bytes que la representación IPv4.

Niveles de índice

Hay tres niveles, o tipos, de indexación: IndexNone, IndexKeys e IndexValues.

IndexNone

Este tipo de índice personalizado no es en realidad un índice. Las entradas de índice personalizado con el nivel IndexNone existen solo para definir y documentar la clave de metadatos. Las entradas IndexNone se pueden usar en índices personalizados de Decoder para imponer un tipo de datos específico para una clave de metadatos a todos los analizadores de un Decoder.

IndexKeys

Este tipo de índice personalizado indica que el índice solo hace un seguimiento de las sesiones que contienen elementos de metadatos con este nombre de clave de metadatos. Sin embargo, no indexa ningún valor único en la base de datos de metadatos para la clave de metadatos.

Los índices en el nivel de clave ocupan mucho menos espacio de almacenamiento, memoria y tiempo de CPU destinado a la administración, pero requieren mucho más trabajo del motor de consultas cuando se ejecutan operaciones query o values que los usan.

Si se usa en una cláusula Where, una clave de metadatos indexada en el nivel de clave solo se puede usar para resolver operaciones como exists o !exists.

IndexValues

Este tipo de índice personalizado mantiene sesiones que contienen cada valor único individual para la clave de metadatos. También se conoce como un "índice completo".

Este tipo de índice se necesita para el procesamiento eficiente de la mayoría de las cláusulas Where y para el uso de esta clave de metadatos como el parámetro fieldName de una llamada values.

Valor máximo

El valor máximo es un parámetro que puede tener un impacto muy significativo en la exactitud y el rendimiento de un índice en el nivel de valor.

A medida que un Decoder analiza paquetes o registros, se le permite crear metadatos de cualquier tipo con cualquier valor. En general, estos elementos de metadatos se crean a partir de datos copiados directamente del paquete o del registro. Por lo tanto, cualquiera puede crear valores de metadatos únicos en respuesta casi a cualquier evento.

El rendimiento del índice depende directamente de la cantidad de valores únicos que ha encontrado para cada clave de metadatos. A medida que la cantidad de valores únicos aumenta, la tasa a la cual se indexan los nuevos metadatos puede disminuir y la velocidad con la cual se completan las consultas baja. Dado que cualquier persona puede influir en la creación de valores de metadatos únicos, es posible que cualquiera afecte el rendimiento del índice.

El parámetro valor máximo limita la cantidad de valores únicos que pueden ingresar al índice. Por lo tanto, un usuario malicioso no puede inundar el sistema con una gran cantidad de valores únicos en un intento por hacer que el sistema NetWitness no funcione.

Es importante configurar un valor máximo en cualquier clave de metadatos cuyo valor pueda recibir la influencia directa de registros o paquetes entrantes.

El valor máximo se aplica solo a valores agregados desde la última operación de guardado del índice.

El límite para la configuración del valor máximo superior varía de una versión a otra y según la cantidad de RAM disponible para el servicio NetWitness NextGen. A partir de 10.3, el límite recomendado para el valor máximo es 5,000,000 para cualquier clave de metadatos. Si hay muchos índices personalizados, es posible que el valor máximo se deba configurar en una cifra inferior.

maxLength

El parámetro de longitud máxima se utiliza exclusivamente en el tipo de metadatos `word` . Debe coincidir con la configuración correspondiente para **/decoder/parsers/config/token.max.length** en el servicio Log Decoder que genera metadatos de token de palabra. El índice utiliza maxLength para interpretar correctamente los términos de búsqueda que se alimentan en la `msearch` función de SDK.

Cambio de nombre de clave

El lenguaje de índice admite el concepto de cambio de nombre de clave. Esta función se utiliza para proporcionar compatibilidad con versiones anteriores para los nuevos nombres de clave con el fin de dejar obsoletos los nombres de clave anteriores y reemplazarlos. Un cambio de nombre se logra mediante la adición de elementos `rename` a la clave. Esto tiene el efecto de indicar que la clave primaria cambia el nombre de la clave en el elemento de cambio de nombre. Por ejemplo, la definición de la clave que aparece a continuación establece una nueva clave denominada `port_src` que cambia el nombre de la clave `tcp.srcport` .

```
<key name="port_src" description="Source Port" format="UInt16"
level="IndexValues">
       <rename name="tcp.srcport"/>
</key>
```

El elemento de cambio de nombre indica a la base de datos que los usos de la clave primaria, en este caso `port_src` , incluirán elementos de metadatos con el tipo port_src y elementos de metadatos con el tipo tcp.srcport. Por lo tanto, los nuevos elementos de metadatos se pueden agregar a la base de datos y consultar mediante `port_src` , y tales consultas devolverán información que también se almacenó anteriormente en tcp.srcport.

El elemento de cambio de nombre acepta un único atributo, `name` , que se refiere a una clave definida anteriormente.

Las claves a las que se hacen referencia los elementos de cambio de nombre deben tener el mismo tipo que la clave primaria.

Las claves a las que se hacen referencia los elementos de cambio de nombre deben tener el mismo nivel de índice que la clave primaria.

Si una clave se redefine en un archivo de índice personalizado y la clave redefinida contiene elementos de cambio de nombre, esos elementos reemplazan a los definidos anteriormente.

Reconstrucción del índice

En una operación normal, los cambios realizados en la configuración del índice para un servicio solo se aplican a los datos nuevos que ingresan a la recopilación. La reconstrucción del índice en todos los datos de la recopilación es un proceso lento debido a que requiere que todo el almacenamiento de la base de datos de metadatos se lea desde el disco.

En la versión 11.0 y superior, es posible reconstruir el índice mientras el servicio está en línea. Los servicios de la versión 11.0 reconstruyen los índices en segundo plano cada vez que el servicio detecta que parte de las bases de datos de sesión y metadatos no está indexada.

Activación del reindexador en segundo plano

El reindexador en segundo plano se activa cada vez que se inicia el servicio. Durante el arranque, el indexador comprueba si hay brechas entre las sesiones indexadas y aquellas que están presentes en la base de datos de sesión y metadatos. Si se encuentran brechas, el reindexador en segundo plano comienza a reindexar la base de datos de sesión y metadatos en el servicio.

Ejemplos de eventos que pueden activar el reindexador en segundo plano:

1. Se produjo una falla de alimentación o una falla general, lo cual dañó el último segmento del índice. Los datos dañados se eliminan en el arranque y esto deja una brecha en el índice.
2. Esto fuerza la eliminación de los datos del índice, ya sea mediante un restablecimiento del índice o la eliminación de los archivos del sistema de archivos.

Control del reindexador en segundo plano

El nodo de configuración `/index/config/reindex.enable` controla la operación del indexador en segundo plano. Si `reindex.enable` se configura en true, el reindexador funcionará la próxima vez que se inicie el servicio. Si `reindex.enable` se configura en false, el reindexador no se iniciará la próxima vez que se inicie el servicio, pero continuará funcionando hasta que este se reinicie.

Algoritmo de reindexación en segundo plano

La operación del indexador en segundo plano es la siguiente:

1. El índice examina los rangos de sesiones que están presentes en el índice y los compara con los rangos de sesiones que tienen metadatos válidos. Las discrepancias entre ambos se consideran brechas.

2. Las brechas en el índice se subdividen en segmentos en función del valor actual de `/index/config/save.session.count`.
3. Para cada segmento que falta, se crea un índice temporal en uno de los directorios que especifica `/index/config/index.dir`. Los segmentos se reindexan en orden numérico inverso. Por lo tanto, las sesiones recopiladas de manera más reciente se indexan en primer lugar.
4. Una vez que el segmento se reindexa por completo, se transfiere a su ubicación válida en el índice en línea. Si el segmento reindexado pertenece al nivel semiactivo, se transfiere al nivel semiactivo.
5. Los datos indexados recientemente aparecen como parte de la recopilación.

Estado del indexador en segundo plano

El nodo de estadísticas `/index/stats/updater.state` indica el estado actual del reindexador en segundo plano. Este nodo indicará `running`, `not running` o `failed`. Si el estado es `failed`, busque más información de diagnóstico en el registro de servicio.

Efectos en la agregación

Los servicios que ejecutan la agregación utilizan el índice para rastrear las sesiones que ya se agregaron. Si el índice no tiene información suficiente para comenzar la agregación, esta se realizará de manera offline hasta que se hayan reindexado los segmentos suficientes. Durante este tiempo, el estado de la agregación para el dispositivo ascendente indicará que está en espera de la agregación.

Cómo forzar una reindexación

Para forzar el índice en un servicio que se reconstruirá:

1. Asegúrese de que `/index/config/reindex.enable` sea true.
2. Restablezca el índice mediante el uso del mensaje `reset` en el servicio. Por ejemplo: `/concentrator/reset index=1` reiniciará el servicio y eliminará todos los archivos del índice.
3. Espere hasta que el servicio se reinicie. Se iniciará la reindexación en segundo plano.
4. Los datos recopilados de manera más reciente estarán disponibles para las consultas tan pronto como el segmento del índice que representa esas sesiones se haya reindexado.

Técnicas de optimización

En este tema se describen las técnicas de optimización para la base de datos de NetWitness Core. La base de datos de NetWitness Core está configurada para trabajar con una amplia gama de cargas de trabajo de manera predeterminada. Sin embargo, como cualquier tecnología de base de datos, su rendimiento puede ser muy sensible tanto a la naturaleza de los datos recopilados, como a la naturaleza de las búsquedas que realiza el usuario en la base de datos.

Límites

Los umbrales son una optimización útil que puede tener un efecto drástico en la velocidad con la que se devuelven resultados a la vista Navegar de NetWitness Suite. Los umbrales se aplican a la llamada `values` . Para obtener más información acerca de la llamada `values` , consulte Consultas .

El umbral define un límite de cuánto se recupera de la base de datos desde el disco con el fin de producir un conteo. Para la mayoría de las consultas, la cantidad de sesiones que coinciden con la cláusula `where` es muy grande. Por ejemplo, si se seleccionan todos los eventos de registro para solo una hora mientras se ejecutan 30,000 eventos por segundo, las coincidencias son 108,000,000 sesiones.

RSA introdujo la función de umbral basándose en la observación de que en la mayoría de los casos en que se requiere un conteo de sesiones, no es necesario tener resultados que sean precisos hasta la última sesión. Por ejemplo, cuando se observan las 20 principales direcciones IP presentes en la última hora, no es muy importante si el informe indica que un valor IP coincidió exactamente con 10,000,000 o 10,000,001 sesiones. El estimado es suficiente. En estos escenarios, podemos realizar un estimado para el valor del conteo devuelto cuando nuestro conteo excede el parámetro de umbral. Cuando se alcanza el umbral, el conteo restante se estima y los resultados se ordenan basados en los conteos estimados, si es necesario.

Cláusulas `where` complejas

La cantidad de tiempo que tarda la base de datos de NetWitness Core en producir un resultado depende de la complejidad de la consulta. Las consultas que se alinean directamente con los índices presentes en los metadatos se pueden resolver rápidamente, pero es muy común escribir consultas que no se puedan resolver rápidamente. En ocasiones, las consultas que no se pueden devolver rápidamente se pueden procesar mediante la base de datos de Core y el índice de manera distinta para producir resultados mucho más satisfactorios para el cliente.

Es útil conocer el *costo* relativo de cada parte de la cláusula `where` . Una cláusula con un costo alto tarda más en ejecutarse. En la siguiente tabla, las operaciones de consulta se ordenan en términos de su costo relativo, del más bajo al más alto.

Operación	Costo
exists, !exists	Constante
= , !=	Logarítmico en términos de la cantidad de valores únicos para la clave de metadatos, lineal en términos de la cantidad de elementos únicos que coinciden con una expresión de rango
< , > , <= , >=	Logarítmico en términos de búsqueda de valores únicos, pero lo más probable es que sea lineal, ya que la expresión coincide con un gran rango de valores
`begins` , `ends` , `contains`	Lineal en términos de cantidad de valores únicos para la clave de metadatos
`regex`	Lineal en términos de cantidad de valores únicos para la clave de metadatos con un costo alto por valor

`AND` y `OR`

Cuando se crea una cláusula `where` , tenga en mente que crear muchos términos con el operador `AND` puede tener una consecuencia benéfica en el rendimiento de una consulta. Siempre que se puedan usar múltiples criterios para filtrar el conjunto de sesiones que coinciden con la cláusula `where` , la carga de trabajo para la consulta es menor. Del mismo modo, cada cláusula `OR` crea un conjunto de sesiones más grande para procesar en cada consulta.

Como regla general, mientras más cláusulas `AND` haya en la consulta, esta se completará con mayor rapidez; por otra parte, mientras más cláusulas `OR` haya en la consulta, esta se completará con mayor lentitud.

Caso de uso: Coincidencia con una subred grande

Es que los usuarios creen consultas que intenten incluir o excluir una subred de clase A. Este tipo de consulta es común debido a que los usuarios intentan incluir o excluir una gran parte de su red interna de su investigación.

Resolver esta consulta con los índices `ip.src` o `ip.dst` solamente es un problema para el motor de consultas. El problema surge a partir del hecho de que una cláusula `where` como esta:

```
ip.src = 10.0.0.0/8
```

En realidad se debe interpretar así:

```
ip.src = 10.0.0.0 || ip.src = 10.0.0.1 || ip.src = 10.0.0.2 || ... ||
ip.src = 10.255.255.255
```

Por lo tanto, el índice podría tener que crear una cláusula `where` con más de 16 millones de términos.

La solución a este problema es usar el Decoder para etiquetar redes de interés comunes mediante reglas de aplicación. Por ejemplo, podría crear elementos de metadatos con una regla de aplicación que se vea así:

```
name=internal rule="ip.src = 10.0.0.0/8" order=3 alert=network
```

Esta regla crea elementos de metadatos en la red de clave de metadatos con el valor interno para cualquier dirección IP en la red 10.0.0.0/8.

La cláusula `where` se podría expresar como:

```
network = "internal"
```

Siempre y cuando haya un índice `value-level` en los metadatos de red, el índice no tiene que expandir esta consulta en nada más complejo y las sesiones coincidentes con la subred deseada coinciden muy rápidamente.

Caso de uso: Coincidencias de subcadena

Usar los operadores `begins` , `ends` , `contains` y `regex` en una cláusula `where` puede ser muy lento si hay una gran cantidad de valores únicos para la clave de metadatos. Cada uno de estos operadores se evalúa de manera independiente contra cada valor único. Por ejemplo, si el operador es `regex` , `regex` se debe ejecutar de manera independiente contra cada valor único.

Para solucionar esto, la estrategia más eficaz es reorganizar los elementos de metadatos de manera que el usuario no tenga que usar una coincidencia de subcadena.

Por ejemplo, suponga que los usuarios intentan encontrar el nombre de host en una URL en algún lugar de la sesión. Los usuarios pueden escribir una cláusula `where` como la siguiente:

```
url contains 'www.rsa.com'
```

En este escenario, es probable que la clave de metadatos `url` contenga un valor único para cada sesión que capturó el Decoder y, por lo tanto, tenga una gran cantidad de valores únicos. En este caso, la operación contains es lenta.

El mejor enfoque es identificar la parte de los metadatos con los cuales se intenta encontrar una coincidencia y mover la coincidencia al analizador de contenido.

Por ejemplo, si se generan metadatos para cada URL, un analizador también podría desglosar la URL en sus componentes constitutivos. Por ejemplo, si el Decoder genera metadatos de URL con el valor `http://www.rsa.com/netwitness` , también podría generar metadatos `alias.host` con el valor `www.rsa.com` . Se pueden realizar consultas mediante:

```
alias.host = 'www.rsa.com'
```

Debido a que el operador de subcadena ya no es necesario, la consulta es mucho más rápida.

Guardados del índice

El índice de Core se subdivide en puntos de guardado, también conocidos como segmentos. Cuando se guarda el índice, todos los metadatos del índice se envían al disco y esa parte del índice se marca como de solo lectura. Los guardados cumplen dos funciones:

- Cada punto de guardado representa un lugar donde el índice se puede recuperar en el caso de una falla en la energía.
- El guardado periódico garantiza que la parte del índice que se está actualizando activamente no crezca más que la memoria RAM.

Los puntos de guardado tienen el efecto de particionar el índice en segmentos independientes y que no se sobreponen. Cuando una consulta debe cruzar múltiples puntos de guardado, debe volver a ejecutar las partes de la consulta y combinar los resultados. En última instancia, esto hace que la consulta se demore más en completarse.

De manera predeterminada, para NetWitness Suite 10.5 e instalaciones posteriores, se realiza un guardado en el índice principal cada vez que se agregan 600,000,000 sesiones a la base de datos. Este intervalo se configura mediante el parámetro de configuración del índice `save.session.count` . Para obtener más información, consulte Nodos de configuración de índices .

Las versiones anteriores de NetWitness Suite, o los sistemas que se actualizaron de versiones de NetWitness Suite anteriores a 10.5, utilizan un programa de guardado basado en tiempo que guarda el índice cada 8 horas. Puede ver el intervalo de guardado actual mediante el uso del editor del programador en la interfaz del usuario de NetWitness Admin para el servicio. La entrada predeterminada tiene la siguiente apariencia:

```
hours=8 pathname=/index msg=save
```

Si ajusta el intervalo, puede controlar la frecuencia con la que se crean los guardados.

Efectos de aumentar el intervalo de guardado

Si se aumenta el intervalo de guardado, se crean puntos de guardado de manera menos frecuente y, por lo tanto, existen menos puntos de guardado. Esto tiene un efecto positivo en el rendimiento de la consulta, ya que se hace menos probable que las consultas recorran segmentos, y cuando sí es necesario recorrer los segmentos, no hay tantos que se deban cruzar.

De todas formas, existen inconvenientes si se aumenta el intervalo de guardado. Primero, es más probable que Concentrator alcance el límite de `valueMax` establecido en cualquiera de los índices. Segundo, se aumenta el tiempo de recuperación en caso de que se produzca un apagado forzado o una falla en la energía. Y tercero, la tasa de agregación puede verse afectada si el segmento del índice crece demasiado como para caber en la memoria.

Efectos de disminuir el intervalo de guardado

Si se disminuye el intervalo de guardado, es posible evitar alcanzar los límites de `valueMax` mientras se mantiene un índice de valores completos para metadatos que contienen una gran cantidad de valores únicos. Si disminuye el intervalo de guardado, no hay un impacto perjudicial en el rendimiento de consultas, ya que se crean más segmentos.

Trabajo con `valueMax`

La limitación de `valueMax` puede ser frustrante para los clientes que deseen indexar todos los metadatos únicos posibles. Desafortunadamente, eso no es posible en el caso general. Existen claves de metadatos que pueden tener datos aleatorios arbitrarios desde cualquier lugar de Internet y no se pueden indexar todos los valores únicos.

Sin embargo, es posible solucionar algunas de las limitaciones de `valueMax` mediante el uso de índices de nivel de clave en lugar de índices de valores. Los índices de nivel clave no reciben la influencia de `valueMax` .

Es posible usar la vista Navegar en una clave de metadatos indexada en el nivel de clave. La base de datos usa índices de nivel de valor en la cláusula `where` cuando es posible, pero se utiliza el escaneo de la base de datos de metadatos para resolver valores únicos para la llamada `values` . Este enfoque funciona bien cuando la cláusula `where` proporciona un filtro eficaz para limitar el alcance de búsqueda a una pequeña cantidad de sesiones, tal vez menos de 10,000.

En casos donde se alcanza `valueMax` , los usuarios pueden realizar un escaneo de la base de datos en sus consultas para asegurarse de que no se hayan descartado valores pertinentes. Esta función es accesible en el cliente de Investigator 9.8 mediante el menú del botón secundario en la vista Navegación. Aunque el escaneo de la base de datos de metadatos tarda bastante, garantiza al cliente que no se está perdiendo nada en sus informes.

Paralelizar cargas de trabajo

Cuando el cliente esté utilizando varios informes, asegúrese de que no esté haciendo uso completo de las opciones de ejecución paralela en Reporting Engine. Del mismo modo, asegúrese de que la cantidad de `max.concurrent.queries` sea adecuada para el hardware.

La vista Navegar de NetWitness Suite tiene la capacidad de ejecutar los componentes de su salida en paralelo, lo cual puede tener un impacto significativo en el rendimiento percibido del servicio NetWitness Core.

Reconstruir el índice

En casos aislados, un servicio Core se puede beneficiar producto de una reconstrucción del índice. Ejemplos:

- El servicio NetWitness Core tiene segmentos de índice creados con una versión muy antigua del producto y no ha transferido ningún dato en más de seis meses.
- El índice se configuró de manera incorrecta y el cliente desea volver a indexar todos los metadatos con una nueva configuración de índice.
- La carga de tráfico en el servicio Core era muy ligera y el intervalo de guardado era grande, lo cual provocó la generación de más segmentos de los necesarios.

En estos casos, una reconstrucción del índice puede proporcionar mejoras de rendimiento. Para hacerlo, debe enviar el mensaje `reset` con el parámetro `index=1` a la carpeta `/decoder` en un Decoder, la carpeta `/concentrator` en un Concentrator o la carpeta `/archiver` en un Archiver.

Tenga en cuenta que una reindexación completa tarda días en completarse en un Concentrator completamente cargado y posiblemente semanas en un Archiver lleno.

Escalamiento de retención

Existen varias formas de mejorar la retención de la base de datos de NetWitness Core. Retención hace referencia al período que cubren los datos almacenados en la base de datos.

El primer paso para analizar la retención es determinar qué parte de la base de datos es el factor limitante en términos de retención. Las bases de datos de paquetes, metadatos y sesiones proporcionan las estadísticas `packet.oldest.file.time` , `meta.oldest.file.time` y `session.oldest.file.time` en la carpeta `/database/stats` para mostrar la antigüedad del archivo más antiguo en la base de datos. El índice proporciona la estadística `/index/stats/time.begin` para mostrar el tiempo de sesión más antiguo que almacena.

Aumentar la retención de paquetes y metadatos

El mecanismo primario para aumentar la retención en estas bases de datos es agregar más almacenamiento. Si no es posible agregar más almacenamiento al servicio NetWitness Core, puede que sea necesario utilizar las opciones de compresión en la base de datos de paquetes y metadatos para disminuir la cantidad de datos que escribe cada base de datos.

Si la retención de metadatos es una preocupación, es posible que desee eliminar el contenido innecesario del Decoder que genera metadatos. Muchos analizadores generan metadatos que el cliente no necesita almacenar a largo plazo.

Aumentar la retención del índice

Normalmente, el índice no tiene más retención que las bases de datos, pero con un índice personalizado complejo, la retención del índice puede ser más corta. Normalmente, el curso de acción más fácil es eliminar los índices de nivel de valor innecesarios de la configuración personalizada o tal vez reemplazar algunos de los índices de nivel de valor predeterminados por índices de nivel de clave.

También es posible escalar el índice mediante la adición de almacenamiento del índice adicional. Sin embargo, el almacenamiento del índice se debe extender solo mediante el uso de unidades de estadio sólido.

Escalar de manera horizontal

A partir de la versión 10.3, los Concentrators y Archivers tienen la capacidad de incorporarse a clústeres mediante la agregación de grupos. La agregación de grupos permite que un solo Decoder alimente sesiones a múltiples Concentrators o Archivers de manera que la carga sea equilibrada. La agregación de grupos permite que la consulta y la carga de trabajo de agregación se dividan en un pool arbitrariamente grande de hardware.

Para obtener más información, consulte el tema “Agregación de grupos” de la *Guía de implementación* .

Cargas de trabajo de grupos

La base de datos de NetWitness Core funciona mucho mejor cuando todos los usuarios del sistema están trabajando dentro de la misma región de la base de datos. Debido a que la base de datos se alimenta de datos del Decoder con un esquema de tipo el primero en entrar, el primero en salir, los datos en la base de datos normalmente se incorporan al clúster juntos según la hora en la que se capturaron y almacenaron. Por lo tanto, la base de datos funciona mejor cuando todos los usuarios están trabajando en el mismo período de datos.

No siempre es posible que todos los usuarios trabajen en el mismo período simultáneamente. La base de datos de NetWitness Core puede manejar ese caso de uso, pero lo hace lentamente porque debe alternar entre tener distintos períodos en la RAM. No es posible tener todos los períodos en la memoria RAM al mismo tiempo. Normalmente menos del 1 % de la base de datos y menos del 10 % del índice caben en la memoria RAM.

Para hacer que NetWitness Suite funcione para el cliente, es importante hacer que este organice sus usuarios en grupos que tienden a trabajar en los mismos rangos de tiempo. Por ejemplo, los usuarios que realizan un monitoreo diario de los datos más recientes pueden ser un grupo de usuarios. Tal vez existe otro grupo de usuarios que realiza consultas más atrás en el tiempo como parte de una investigación. Y tal vez otro grupo de usuarios realiza informes de períodos largos. Intentar servir a todos los grupos desde una base de datos simple puede llevar a frustración y esperas largas para que se produzcan resultados. Sin embargo, si los distintos casos de uso se pueden distribuir entre distintos elementos de hardware de Concentrator, el rendimiento percibido puede ser mucho mejor. En este caso, puede ser beneficioso utilizar más servicios de Concentrator con menos RAM y potencia de CPU que un Concentrator simple, grande y costoso diseñado para cumplir todas las necesidades.

Ventana de caché

Considere esta secuencia de eventos:

1. A las 9:00 h, el usuario “kevin” inicia sesión en un servicio Concentrator y solicita un informe sobre la última hora de tiempo de recopilación.
2. Concentrator recupera informes para el rango de tiempo entre las 8:00 h y las 9:00 h.
3. A las 09:02 h, el usuario “scott” inicia sesión en el mismo Concentrator y además solicita un informe sobre la última hora de tiempo de recopilación.
4. Concentrator recupera informes para el rango de tiempo entre las 8:02 h y las 9:02 h.

Tenga en cuenta que incluso cuando ambos usuarios observaban rangos de tiempo que se acercaban, el trabajo que realiza Concentrator para producir informes para Kevin no se podía volver a enviar a Scott, ya que los rangos de tiempo eran levemente distintos. Concentrator tuvo que volver a calcular la mayoría de los informes para Scott.

La configuración `cache.window.minutes` en el nodo `/sdk` le permite optimizar esta situación. Cuando un usuario inicia sesión, el punto en el tiempo que representa los datos más recientes para la recopilación solo avanza en incrementos de la cantidad de minutos en esta configuración.

Por ejemplo, suponga que `/sdk/config/cache.window.minutes` es 10\. La reevaluación de la acción anterior cambia la secuencia de eventos.

1. A las 9:00 h, el usuario “kevin” inicia sesión en un servicio Concentrator y solicita un informe sobre la última hora de tiempo de recopilación.
2. Concentrator recupera informes para el rango de tiempo entre las 8:00 h y las 9:00 h.
3. A las 09:02 h, el usuario “scott” inicia sesión en el mismo Concentrator y además solicita un informe sobre la última hora de tiempo de recopilación.
4. Concentrator recupera informes para el rango de tiempo entre las 8:00 h y las 9:00 h.

5. A las 09:10 h, el usuario “scott” vuelve a cargar los informes para la última hora del tiempo de recopilación.
6. Concentrator recupera informes para el rango de tiempo entre las 8:10 h y las 9:10 h.

El informe devuelto en el paso 3 queda en la ventana de caché, de modo que se devuelve instantáneamente. Esto le da a Scott la impresión de que Concentrator es muy rápido.

De esta forma, una configuración más amplia de `cache.window` mejora el rendimiento percibido, con el costo de introducir pequeños retrasos hasta que los últimos datos estén disponibles para su búsqueda.

Límites de tiempo

Cuando se ejecuta una consulta en la base de datos de NetWitness Core durante un período muy largo, el servicio principal dedica más y más tiempo de CPU y RAM a esa consulta con el fin de hacer que se complete más rápido. Esto puede tener un impacto perjudicial en otras consultas y agregaciones. Para evitar que los usuarios con menos privilegios utilicen más que su parte de los recursos del servicio Core, es útil poner límites de tiempo en las consultas que ejecutan los usuarios normales.

Apéndice A: Estadísticas

En este tema se describen estadísticas usadas para monitorear la operación del sistema. Los servicios Core proporcionan una cantidad muy grande de estadísticas para el monitoreo de la operación del sistema. Algunas de ellas son útiles para el rendimiento de monitoreo, mientras que otras existen para monitorear la operación del sistema o para fines de depuración.

Estadísticas en /database/stats

En la siguiente tabla se muestra el significado de las estadísticas en **/database/stats** .

Estadísticas	Significado
`meta.bytes` , `packet.bytes` , `session.bytes`	El tamaño total de los datos (en bytes) almacenados en cada base de datos
`meta.first.id` , `packet.first.id` , `session.first.id`	La primera ID de metadatos, ID de paquete e ID de sesión, respectivamente, almacenadas en la base de datos
`meta.last.id` , `packet.last.id` , `session.last.id`	La última ID de metadatos, ID de paquete e ID de sesión, respectivamente, almacenadas en la base de datos
`meta.oldest.file.time` , `packet.oldest.file.time` , `session.oldest.file.time`	La fecha de creación del archivo más antiguo en cada base de datos
`meta.rate` , `packet.rate` , `session.rate`	El conteo de la cantidad de objetos de metadatos, paquetes y sesiones agregados a cada base de datos en el último segundo
`meta.total` , `packet.total` , `session.total`	La cantidad total de objetos de metadatos, paquetes y sesiones dentro de cada base de datos
`meta.volume.bytes` , `packet.volume.bytes` , `session.volume.bytes`	El tamaño de volumen total aproximado (en bytes) de todos los directorios utilizados por cada base de datos
`meta.free.space` , `packet.free.space` ,	El espacio no usado total aproximado (en bytes) en todos los directorios usados por cada base de datos

`session.free.space`

Estadísticas en /index/stats

En la siguiente tabla se muestra el significado de las estadísticas en **/index/stats** .

Estadísticas	Significado
`checkpoint.page` , `checkpoint.summary`	Los últimos objetos almacenados la última vez que se creó un guardado de índice (depuración)
`index.bytes`	Una medida aproximada de cuánto espacio en disco requieren los archivos de índice
`index.last.load.time`	El registro de fecha y hora en que se cargó la configuración del índice actual desde los archivos de configuración del índice
`memory.used`	Una medida aproximada de cuánta memoria ocupa el índice
`page.first.id` , `summary.first.id`	La primera página y el objeto de resumen almacenados en el índice (depuración)
`page.last.id` , `summary.last.id`	La última página y el objeto de resumen almacenados en el índice (depuración)
`page.total` , `summary.total`	Cantidad de páginas y resúmenes en el índice(depuración)
`session.first.id`	El ID de la primera sesión indexada
`session.last.id`	El ID de la última sesión indexada
`sessions.since.save`	La cantidad de sesiones que contiene el segmento de índice actual
`values.added`	La cantidad de valores únicos agregados al segmento de índice actual
`slices.total`	La cantidad de segmentos en el índice.
`time.begin`	Los metadatos de hora más antiguos indexados
`time.end`	Los metadatos de hora más recientes indexados
`updater.state`	El estado del reindexador en segundo plano

Estadísticas en /sdk/stats

En la siguiente tabla se muestra el significado de las estadísticas en **/sdk/stats**

Estadísticas	Significado
`cache.window.time.begin`	El principio de la hora actual impuesto por cache.window.minutes
`cache.window.time.end`	La finalización de la hora actual impuesta por cache.window.minutes
`queries.active`	La cantidad de consultas actualmente en ejecución en el índice
`queries.queued`	La cantidad de consultas esperando su ejecución
`values.calls`	La cantidad de llamadas hechas a la función “values” desde que se inició el proceso
`values.calls.cached`	La cantidad de llamadas hechas a la función “values” que resolvió la caché de resultados de la llamada values

Estadísticas por consulta

Las operaciones de SDK, como query y values, proporcionan información acerca de su estado de ejecución en `/sdk/config/stats/queries/<handleid>` , donde `<handleid>` es un identificador único para la operación de consulta.

En la siguiente tabla se muestra el significado de las estadísticas por consulta.

Estadísticas	Significado
channel.path	Esta estadística proporciona un vínculo al canal de conexión mediante el cual se comunica la operación. Este canal se usa para comunicar resultados al cliente.
`query.type`	El tipo de operación que se está ejecutando, como consultas o valores
`query`	El conjunto completo de parámetros entregados a la consulta
`query.progress`	El porcentaje de ejecución de consultas completadas
`query.status`	Un mensaje que describe qué etapa de la ejecución de la consulta está ocurriendo actualmente

`running.since`	La hora en la que se inició la ejecución de la consulta
`user`	El nombre de usuario que ejecutó la consulta

Apéndice B: Inspección del índice

El índice de la base de datos de NetWitness Core tiene una función de depuración incorporada denominada `inspect` que proporciona información detallada sobre la composición de sus índices. La función de inspección del índice se localiza en `/index/inspect` en cada árbol de configuración de los servicios principales. En los servicios que no tienen realmente un índice, como Broker, la función `/index/inspect` no está presente.

Parámetros

Opciones

Tipo: Cadena Este parámetro se puede configurar en el valor `all-slices` para recopilar información de inspección sobre cada segmento del índice. Si no se configura, se devuelve información sobre el segmento actual creado de manera más reciente.

La recopilación de información sobre todos los segmentos puede tardar mucho tiempo si hay muchos segmentos en el índice.

Respuesta

La inspección devuelve muchas filas de pares de valores de claves que representan el estado del índice.

Resumen del segmento

La primera fila que se devuelve para cada segmento es un resumen con los siguientes valores.

`session1`	El ID de la primera sesión indexada en el segmento
`session2`	El ID de la última sesión indexada en el segmento
`meta1`	El ID de los primeros metadatos de la primera sesión indexada en el segmento
`meta2`	El ID de los últimos metadatos de la última sesión indexada en el segmento

Resumen por índice

Se devolverán filas de resumen por índice para cada índice. Solo se informan índices en el nivel de valores.

`key`	El nombre de la clave de metadatos para el índice
`pathname`	La ruta en el disco a este índice

`values`	La cantidad de valores únicos que se almacenan en este índice
`summaries`	La cantidad de entradas de resumen que ocupa este índice en el archivo summary.db
`pages`	La cantidad de entradas de página que ocupa este índice en el archivo page.db
`sessions`	La cantidad de sesiones que tenían un valor que se insertó en este índice
`size`	Los valores de metadatos de "tamaño" acumulativos para todas las sesiones que insertaron un valor en este índice
`packets`	El conteo de paquetes acumulativo para todas las sesiones que insertaron un valor en este índice
`summary1`	El ID del primer resumen que usó este índice
`summary2`	El ID del último resumen que usó este índice
`session1`	El ID de la primera sesión al cual hace referencia este índice
`session2`	El ID de la última sesión al cual hace referencia este índice

Pie de página del resumen del segmento

La última fila de cada informe de inspección contiene estadísticas acumulativas para todos los índices del segmento.

`totalKeys`	La cantidad de tipos de metadatos indexados
`totalValues`	La cantidad de valores únicos que rastrean todos los índices en este segmento
`totalMemory`	Un total aproximado de la memoria necesaria para abrir este segmento del índice