RSA NETWITNESS® SUITE

Tuningleitfaden für die Core-Datenbank

für Version 11.0

Kontaktinformationen

Der RSA-Link unter https://community.rsa.com enthält eine Wissensdatenbank, in der allgemeine Fragen beantwortet und Lösungen für bekannte Probleme, Produktdokumentationen, Communitydiskussionen und Vorgangsmanagement bereitgestellt werden.

Marken

Eine Liste der RSA-Marken finden Sie unter germany.emc.com/legal/emc-corporation-trademarks.htm#rsa.

Lizenzvereinbarung

Diese Software und die zugehörige Dokumentation sind Eigentum von EMC und vertraulich. Sie werden unter Lizenz bereitgestellt und dürfen nur gemäß den Bedingungen der betreffenden Lizenz und unter Einschluss des untenstehenden Copyright-Hinweises verwendet und kopiert werden. Diese Software und die Dokumentation sowie alle Kopien dürfen anderen Personen nicht überlassen oder auf andere Weise zur Verfügung gestellt werden.

Dabei werden keine Ansprüche oder Eigentumsrechte an der Software oder Dokumentation oder Rechte an geistigem Eigentum daran übertragen. Die unberechtigte Nutzung oder die Vervielfältigung dieser Software und der Dokumentation kann zivil- und/oder strafrechtlich verfolgt werden.

Diese Software kann ohne Vorankündigung geändert werden und sollte nicht als Verpflichtung seitens EMC ausgelegt werden.

Drittanbieterlizenzen

Dieses Produkt kann Software enthalten, die von anderen Anbietern als RSA entwickelt wurde. Der Text der Lizenzvereinbarungen, die sich auf Drittanbietersoftware in diesem Produkt beziehen, ist auf der Produktdokumentationsseite auf RSA Link verfügbar. Mit der Verwendung dieses Produkts verpflichtet sich der Benutzer zur uneingeschränkten Einhaltung der Bedingungen der Lizenzvereinbarungen.

Hinweis zu Verschlüsselungstechnologien

Dieses Produkt kann Verschlüsselungstechnologie enthalten. In vielen Ländern ist die Verwendung, das Importieren oder Exportieren von Verschlüsselungstechnologien untersagt. Die aktuellen Bestimmungen zum Verwenden, Importieren und Exportieren sollten beim Verwenden, Importieren und Exportieren dieses Produkts eingehalten werden.

Verteilung

EMC ist der Ansicht, dass die Informationen in dieser Veröffentlichung zum Zeitpunkt der Veröffentlichung korrekt sind. Diese Informationen können jederzeit ohne vorherige Ankündigung geändert werden.

Februar 2018

Inhalt

Einführung in die NetWitness-Core-Datenbank

Dieses Thema bietet eine Übersicht über die NetWitness-Core-Datenbank. Die NetWitness-Core-Services enthalten eine proprietäre Datenbank, die speziell für die Verwendung mit NetWitness Suite-Produkten entwickelt wurde. Sie hat nur wenig Ähnlichkeit mit den herkömmlichen relationalen Datenbanken und basiert nicht auf einer handelsüblichen Datenbanktechnologie. Daher ist es für viele Benutzer ein intensiver Lernprozess, bis sie verstehen, wie die Core-Datenbank funktioniert und wie sie sie optimal nutzen können. Der Zweck dieses Leitfadens besteht darin, den NetWitness Suite-Benutzern die Funktionsweise der Datenbank zu verdeutlichen und zu erläutern, wie sie die Funktionen optimal ausschöpfen können.

Als Systemadministrator können Sie diese Informationen nutzen, um die Planung Ihrer NetWitness Suite-Bereitstellung zu erleichtern und sie auf die optimale Performance abzustimmen. Als Analyst können Sie diesen Leitfaden verwenden, um Ihre Analyse so zu strukturieren, dass Berichte schneller zurückgegeben werden. Als Inhaltsentwickler können Sie diesen Leitfaden verwenden, um Inhalte zu schreiben, die vom Datenbanksystem effizient verarbeitet werden.

In diesem Leitfaden behandelte NetWitness Suite-Produkte

In diesem Leitfaden werden die Funktionen der NetWitness Suite 11.0 behandelt. Die folgenden NetWitness Suite-Komponenten enthalten die Core-Datenbank:

- Concentrator
- Archiver
- Decoder
- Log Decoder
- Workbench

Häufig verwendete Begriffe

In diesem Abschnitt werden Definitionen für die in diesem Dokument verwendeten Begriffe erklärt. Die Begriffe werden in der Reihenfolge genannt, in der sie im NetWitness Suite-System vorkommen:

- **Paket-DB** : Die Paketdatenbank enthält die erfassten Rohdaten. Auf einem Decoder enthält die Paketdatenbank Pakete, wie sie vom Netzwerk erfasst wurden. Log Decoder verwenden die Paketdatenbank zum Speichern von Rohdatenprotokollen. Die in der Paketdatenbank

gespeicherten Rohdaten können über eine Paket-ID aufgerufen werden, diese ID wird jedoch normalerweise für den Anwender nicht angezeigt.

- **Paket-ID** : Eine Nummer, anhand der ein Paket oder ein Protokoll in einer Paketdatenbank eindeutig identifiziert wird.
- **Meta-DB** : Die Metadatenbank enthält Informationen, die von einem Decoder oder Log Decoder aus dem Rohdatenstream extrahiert werden. Parser, Regeln oder Feeds können Metaelemente erzeugen.
- **Meta-ID** : Eine Nummer, anhand der ein Metaelement in der Metadatenbank eindeutig identifiziert wird.
- **Metaschlüssel** : Ein Name, der zum Klassifizieren des Typs des jeweiligen Metaelements dient. Allgemeine Metaschlüssel sind u. a. ip.src, time oder service.
- **Metawert** : Jedes Metaelement enthält einen Wert. Der Wert entspricht dem vom Parser, vom Feed oder von der Regel erzeugten Wert.
- **Sitzungs-DB** : Die Sitzungsdatenbank enthält Informationen, die das Paket und die Metaelemente in Sitzungen miteinander verknüpft.
- **Sitzung** : Für einen Paketdecoder stellt eine Sitzung einen einzelnen logischen Netzwerkdatenstream dar. Zum Beispiel ist eine TCP/IP-Verbindung eine Sitzung. Für einen Log Decoder ist jedes Protokollereignis eine Sitzung. Jede Sitzung enthält die Verweise auf alle Paket-IDs und Meta-IDs, die sich auf die Sitzung beziehen.
- **Sitzungs-ID** : Eine Nummer, anhand der Sitzungen in der Sitzungs-DB eindeutig identifiziert werden.
- **Index** : Der Index ist eine Sammlung von Dateien, anhand dessen Sitzungs-IDs mithilfe von Metawerten gesucht werden können.
- **Core-Datenbank** : Diese Datenbank ist die Kombination der Paket-, Meta-, Sitzungs- und Indexdatenbank.

Für Syntaxdefinitionen werden in diesem Dokument EBNF -Grammatikdefinitionen verwendet.

Verlauf der NetWitness-Core-Datenbank

NetWitness (jetzt RSA) hat die Core-Datenbank zur Verwendung in Paketerfassungssystemen entwickelt. Frühzeitig im Entwicklungsprozess von NetWitness haben Entwickler festgestellt, dass die vorhandenen Datenbanktechnologien nicht mit den hohen Aufnahmeraten, die zur Erfassung von vollen Paketen gehören, Schritt halten können. Moderne Datenbanktechnologien konnten nicht annähernd mit der Erfassung der großen Anzahl von pro Sekunde empfangenen Sitzungen Schritt halten, geschweige denn, mit der Sortierung aller Pakete. Aufgrund des Datenvolumens muss der Paketspeicher genauso schnell verworfen und wiederverwendet werden, wie er verarbeitet wird. Dies stellte zum damaligen Zeitpunkt einen Schwachpunkt von Datenbanken dar. Daher hat NetWitness eine Datenbank entwickelt, die aus Paket-, Sitzungs- und Metadatenbanken besteht.

Damit die Analysefunktionen von NetWitness Investigator bereitgestellt werden können, wurde der NetWitness-Datenbank ein Metaindex hinzugefügt. Der Index verfolgte dieselben Ziele bezüglich Design wie die ursprünglichen Datenbanken. Er wurde so konzipiert, dass er eine sehr hohe Einfügerate in eine hohe Anzahl von umfangreichen Indizes unterstützt.

Der Index wurde im Laufe der Jahre beträchtlich weiterentwickelt. Frühere Versionen des Index konnten nur zusammenfassende Schätzungen abgeben, wie viele eindeutige Metawerte in der Metadatenbank vorhanden waren. Andere Versionen hatten große Mühe, die akzeptable Abfrageperformance zu erfüllen. So ließ sich zum Beispiel in NetWitness 9.0 die Berichtsdauer häufig in Minuten anstatt in Sekunden messen. Die aktuelle Indexversion wurde vom NetWitness 9.0-Index abgeleitet und beträchtlich weiterentwickelt, um die Performanceerwartungen zu erfüllen und neue Funktionen hinzufügen zu können.

Stärken und Schwächen der Core-Datenbank

Stärken:

- Kontinuierlich hohe Einfügeraten, ohne dass Ausfallzeiten für Masseneinfügungen erforderlich sind
- Akzeptable Abfrageperformance bei hohen Einfügeraten
- Automatische Bereinigung und Rollover von alten Daten mit minimaler Fragmentierung
- Sehr hohe Anzahl von Metawerteindizes: auf einem Concentrator sind standardmäßig mehr als 100 Indizes aktiviert.
- Funktion zum Skalieren von Datenbanken in Petabyte-Größe und Indizes in Terabyte-Größe auf einem einzigen Node

- Durch die Verwendung von Metaschlüssel/Wert-Paaren ist die Datenbank äußerst flexibel für die Speicherung von beliebigen Metaelementen innerhalb einer Sitzung. Daher kann eine Sitzung für die Darstellung von nahezu allen Datensatztypen verwendet werden.

Schwächen:

- Die Abfragefunktion ist begrenzt und erfolgt auf niedriger Stufe.
- Das Schema der Paket-, Meta- und Sitzungs-DB ist festgelegt und sämtliche Anpassungen werden über benutzerdefinierte Metaschlüssel und -werte vorgenommen.
- Die Datenbank bietet keine Garantien zur Transaktionsunteilbarkeit, wie Sie sie von einer SQL-Datenbank erwarten würden.

Grundlegende Datenbankkonfiguration

Dieses Thema behandelt die allgemeinen Einstellungen der Datenbankkonfiguration von NetWitness-Core-Services. Informationen zur Konfiguration der Core-Services durch Bearbeiten der Konfigurationsdateien finden Sie unter „Servicekonfigurationseinstellungen“ im *Leitfaden zu den ersten Schritten mit Hosts und Services* .

Dieses Dokument setzt Vorkenntnisse im Bereich der Konfigurationsanpassung eines NetWitness-Core-Services seitens des Lesers voraus. Sie können dieses Dokument als Leitfaden verwenden, wenn Sie mit einem der Mechanismen zur Bearbeitung der Konfigurationsstruktur von Core-Services vertraut sind. Beispiele für diese Mechanismen sind die Ansicht „Durchsuchen“ auf den Administrationsseiten innerhalb der NetWitness Suite-Benutzeroberfläche oder die REST-Benutzeroberfläche, auf die in jedem Service durch einen Webbrowser zugegriffen werden kann.

Hilfe innerhalb des Core-Services

Jedes Konfigurationselement innerhalb eines Core-Services verfügt über eine integrierte Beschreibung der Funktionsweise des Elements. Sie können die Hilfe-Informationen anzeigen, indem Sie Ihre Maus über das Konfigurationselement in der Ansicht Durchsuchen bewegen. Jedes Konfigurationselement zeigt zudem an, ob dieses ohne Neustart verändert werden kann oder ob ein Neustart erforderlich ist, um die Änderung zu übernehmen.

Entwickler, die REST API verwenden, können auf den Hilfetext jedes einzelnen Konfigurationselement zugreifen, indem Sie die Meldung `help` an den Konfigurationsknotenpfad senden.

Paket-, Meta- und Sitzungsspeicher

Jede der Paket-, Meta- und Sitzungsdatenbanken wird durch den Ordner `/database/config` auf jedem NetWitness-Core-Service konfiguriert. Jede Datenbank verfügt über einen konfigurierbaren Parameter, in den der Core-Service Daten speichert. Paket-, Meta- und Sitzungsdatenbanken folgen einem wiederkehrenden Muster für alle ihre Konfigurationseinträge. Konfigurationselemente einer Paketdatenbank beginnen mit dem Präfix `packet` , Metadatenbanken mit dem Präfix `meta` und die Konfigurationselemente der Sitzungsdatenbank beginnen mit dem Präfix `session` .

Indexspeicher

Die Indexkonfiguration wird in dem Ordner `/index/config` auf jedem Core-Service gespeichert.

Themen

Tiered-Datenbankspeicher

In diesem Thema werden Tiered-Datenbankspeicher beschrieben und Empfehlungen für Hot-, Warm- und Cold-Tier-Storage gegeben.

Seit der Version 10.4 kann der Archiver-Service so konfiguriert werden, dass er Tiered Storage verwenden kann. Die Idee hinter Tiered Storage ist es, die aktuellen Daten auf einen Hot-Tier zu stellen, den am schnellsten verfügbaren Speicher auf dem Archiver.

Alle Services verwenden standardmäßig den Hot-Tier.

Der nächste Tier ist der Warm-Tier. Das ist normalerweise ein preiswerterer und langsamerer Speicher, wie etwa ein Network Attached Storage (NAS). Der Warm-Tier enthält ältere Daten; wie alt sie sind, hängt davon ab, wie viel Speicher auf dem Hot-Tier allokiert ist, sowie von der durchschnittlichen Aufnahmerate. Wenn der Hot-Tier die maximale Auslastung erreicht, werden normalerweise die ältesten Daten vom Hot-Tier auf den Warm-Tier verschoben. Wenn alles richtig konfiguriert ist, geschieht dies automatisch und für den Anwender unsichtbar. Abfragen und Datenzugriff geschehen automatisch, egal auf welchem Tier (Hot oder Warm) die Daten liegen. Allerdings kann es beim Zugriff auf Daten auf dem Warm-Tier im Vergleich zu dem Hot-Tier zu Performanceeinbußen kommen, da der Zugriff auf den Warm-Tier typischerweise langsamer ist.

Zusätzlich zum Hot- und Warm- gibt es auch noch einen Cold-Tier. Der Cold-Tier wird nur als Staging-Bereich für das Offline-Backup verwendet. NetWitness-Core-Services greifen nicht auf Daten auf dem Cold-Tier zu. NetWitness-Core-Services verschieben die ältesten Daten auf den Cold-Tier und betrachten sie als aufgegeben (der Service greift nicht länger auf die Daten zu). Diese Daten können dann langfristig gespeichert werden. Sie können zum Beispiel auf Band gesichert werden, damit sie bei Bedarf Monate oder Jahre später wiederhergestellt werden können. Das Sichern und nachfolgende Entfernen der Daten auf dem Cold-Tier muss außerhalb der NetWitness-Services über Skripte oder andere Prozesse erfolgen.

Wenn der Cold-Tier voll wird, weil externe Prozesse die Daten nicht rechtzeitig entfernen, wird der NetWitness-Core-Service die Aufnahme neuer Daten möglicherweise beenden, bis das Problem behoben ist.

Wenn Daten auf den Cold-Tier verschoben werden, empfiehlt RSA, das Verzeichnis auf demselben Mount-Punkt zu lassen, von dem es verschoben wird. Wenn daher die Daten von dem Warm-Tier kommen, ist es aus Performancegründen viel besser, das Cold-Tier-Verzeichnis auf dasselbe Dateisystem zu stellen. Der Grund dafür ist, dass der Service versucht, die Datei und das Verzeichnis einfach auf den Cold-Tier zu verschieben – ein Vorgang, der nahezu sofort auf demselben Dateisystem abgeschlossen ist. Wenn das Verschieben fehlschlägt, besteht die Sicherung darin, die Daten auf den Cold-Tier zu verschieben, was länger dauert und zusätzliche I/O-Konkurrenz auf dem Tier verursacht, von dem die Daten kopiert werden.

Archiver

Die Tiers der Speicherfunktionen werden vom Archiver verwendet. Sie können Archiver so konfigurieren, dass nur der (standardmäßige) Hot-Storage verwendet wird, Hot und Warm, oder alle drei (Hot, Warm und Cold). Alle Services müssen Hot verwenden. Sie können keinen Service so konfigurieren, dass er nur Warm verwendet. Die Daten fließen von Hot zu Warm und schließlich zu Cold. Sie können Warm auch überspringen und von Hot zu Cold gehen. Wenn Cold-Storage (offline) nicht konfiguriert ist, werden die ältesten Daten auf dem letzten konfigurierten Tier gelöscht. Dies ist bislang das Standardverfahren.

Die typische Archiver-Bereitstellung stellt alle Datenbanken auf unbegrenzte Größe ein (packet.dir, meta.dir, session.dir, index.dir und optional die Warm-Tier-Varianten). Das heißt, dass die Größenangabefunktion entweder ausgeschaltet oder auf Null eingestellt wird. So können die Datenbanken und der Index unbegrenzt wachsen. Anstatt dass jede Datenbank ihre eigene Größe verwaltet und Daten nur dann verlagert, wenn jede einzelne Datenbank ihre konfigurierte Größe überschreitet, verlagert Archiver alles zusammen mithilfe des Befehls `/index sizeRoll` . So können die Datenbanken und der Index die Daten gemeinsam verlagern. Weitere Informationen über sizeRoll finden Sie unter „Asynchrones Rollover“ in Rollover .

Archiver wird typischerweise so konfiguriert, dass Index-, Sitzungs-, Metadaten- und Paket (Protokoll)-Datenbank auf dem gleichen Volume liegen, anstatt auf mehreren Volumes wie ein Concentrator oder Decoder. Obwohl dies potenziell zu mehr I/O-Konkurrenz führen kann, wenn gleichzeitige Lesevorgänge über mehrere Datenbanken geschehen, maximiert es auch die allgemeine Aufbewahrung. Da alle Datenbanken auf demselben Volume liegen, sind sie so konfiguriert, dass sie Daten gemeinsam verlagern, wodurch die Verwaisung von Daten minimiert wird. Decoder und Concentrator werden für maximale I/O-Geschwindigkeit konfiguriert, können aber unter Fehleinschätzungen der richtigen Dimensionierung der Volumes leiden.

Beispiel: Wenn die Sitzungsdatenbank zu groß ist, hat sie möglicherweise ausreichend Speicherplatz für sechs Monate Aufbewahrung, während die Metadaten- und Indexdatenbank nur Platz für vier Monate Aufbewahrung hat. Weil die Sitzungs-, Metadaten- und Indexdatenbank miteinander verbunden sind, definiert die kürzeste Aufbewahrungsfrist der drei die allgemeine Aufbewahrungsfrist (in diesem Fall vier Monate). Die Aufbewahrung einzelner Datenbanken ist meist von Faktoren betroffen, die wir nicht unter Kontrolle haben, wie etwa erfasster Datenverkehr, erzeugte Metadaten (Parser, Feeds, Regeln) und Filterung. Die Größe der Datenbanken lässt sich durch eine einfache Konfigurationsänderung neu festlegen. Aber dafür müssen normalerweise auch Änderungen auf Hardware- und Dateilevel vorgenommen werden, um Partitionen zu ändern, wodurch die dynamische Größenänderung kompliziert wird. Archiver vermeidet diese Probleme, indem ein einziges Volume für alles verwendet wird, wofür eine etwas geringere I/O-Geschwindigkeit in Kauf genommen wird.

Manifeste

Dieses Thema enthält eine Beschreibung der Manifestdateien und ein Beispielmanifest für eine Meta-DB-Datei. Außerdem wird das Durchsuchen von Manifesten beschrieben und ein Beispiel für eine Manifestsuche bereitgestellt.

Manifestdateien werden mit allen Sitzungen, Metadaten, Paket-(Protokoll)-DB-Dateien und Indexsliceverzeichnissen erstellt. Eine Manifestdatei beschreibt verschiedene wichtige Informationen zu den Daten, auf die sie sich bezieht. Manifestdateien werden in Form von JSON-Datensätzen geschrieben. Manifestdateien werden zusammen mit den Daten, die sie darstellen, von einem Tier zum nächsten verschoben. Wenn die zugehörigen Daten gelöscht werden, wird die Manifestdatei außer im folgenden Sonderfall ebenfalls gelöscht. Wenn für den Service in der Datei `/database/config/manifest.dir` ein gültiges Verzeichnis konfiguriert ist, wird beim Löschen der Manifestdaten eine Kopie der Manifestdatei in dem Verzeichnis abgelegt, auf das in der Datei `manifest.dir` verwiesen wird. (Wenn das Verzeichnis nicht vorhanden ist, wird es erstellt.) Dadurch wird die in NetWitness Suite bereitgestellte Funktion der historischen Manifestsuche aktiviert.

Durch diesen Prozess sollen historische Manifestdateien über Jahre hinweg an einer Stelle aufbewahrt werden, damit sie für Offline-Abfragen zur Verfügung stehen. Wenn ein Service mehrere Jahre lang ausgeführt wird, kann dies natürlich zu Hunderttausenden von Dateien führen. Das sollte allerdings kein Problem darstellen, da der Service Dateien zur Einsparung von Speicherplatz automatisch in ein einziges Archiv komprimiert, wenn sie zu zahlreich werden. Manifestdateien sind sehr klein und lassen sich gut komprimieren.

Beispielmanifest (`meta-000000023.nwmdb.manifest`) für eine Meta-DB-Datei:

```
{
    "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
```

Die wichtigsten Felder in diesem Beispielmanifest sind `fileTime` , `time1` und `time2` . Alle drei Felder werden in POSIX-Zeit angegeben. `time1` und `time2` stellen die Anfangs- und Endzeit der in der Meta-DB-Datei `meta-000000023.nwmdb` festgehaltenen Metadaten dar. Das Feld `fileTime` gibt insbesondere immer die Zeit an, zu der die Datei erstellt wurde (und nicht den Zeitpunkt der letzten Änderung). `time1` und `time2` entsprechen dem minimalen und maximalen Bereich der analysierten Daten in der Meta-DB-Datei. Beim Ausführen von zeitbasierten Verlaufssuchen werden die Felder `time1` und `time2` (sofern vorhanden) dem Feld `fileTime` vorgezogen. Manifestdateien für die anderen Datenbanken und den Index enthalten einige andere Felder, aber alle Manifestdateien umfassen genügend Informationen zum Ausführen zeitbasierter Abfragen.

Durchsuchen von historischen Manifesten

Wenn Manifeste in dem Verzeichnis gesammelt werden, auf das in der Datei `manifest.dir` verwiesen wird, wird davon ausgegangen, dass die Daten, auf die sich die Manifeste beziehen, in den Cold Tier kopiert und letztendlich auf einem Offlinespeichermedium gesichert wurden. Da der Service weiterhin auf die historischen Manifeste zugreifen kann, können zeitbasierte Abfragen für Offlinedaten durchgeführt werden. So kann ermittelt werden, welche Daten für einen bestimmten Zeitraum wiederhergestellt werden müssen.

Sie können Manifeste mithilfe des Befehls `/database manifest` durchsuchen:

```
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
```

Suchbeispiel:

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

Die Suche gibt alle Manifeste zurück, die der Abfrage entsprechen:

```
[ 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 ]
```

Anhand der zurückgegebenen Ergebnisse kann abgeleitet werden, welche Dateien für den angegebenen Zeitraum aus dem Backup wiederhergestellt werden sollen. Für NetWitness Suite 10.4 und später kann der Workbench-Service verwendet werden, um die wiederhergestellten Dateien aufzunehmen und eine Abfrageoberfläche für die wiederhergestellten Daten mithilfe von Sammlungen bereitzustellen.

Die Einrichtung des Workbench-Services geht über den Rahmen dieses Dokuments hinaus. Weitere Informationen finden Sie unter „Konfigurieren von Datenbackup und -wiederherstellung“ im *Konfigurationsleitfaden zu Archiver* .

Erweiterte Datenbankkonfiguration

In diesem Thema werden die erweiterten Konfigurationsoptionen der NetWitness-Core-Datenbank erläutert.

Die Konfigurationsoptionen der NetWitness-Core-Datenbank können sich von einer Version zur nächsten ändern. Viele der Configuration Items ändern sich jedoch nur selten. Die zugehörige Dokumentation finden Sie hier. Diese Liste ist nicht erschöpfend, da bei jeder Version neue Funktionen hinzugefügt werden, für die ggf. auch neue Configuration Items erforderlich sind. Die aktuelle Dokumentation finden Sie in der integrierten Hilfefunktion des NetWitness-Core-Services.

Themen

Datenbankkonfigurations-Nodes

In diesem Thema werden Datenbankkonfigurations-Nodes beschrieben. Die folgenden Datenbankkonfigurations-Nodes gehören zu den erweiterten Datenbank-Konfigurationselementen der NetWitness Core-Datenbank, die nur selten geändert werden.

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

Dies ist der primäre Konfigurationseintrag für jede Datenbank (auch Hot-Tier genannt). Damit wird gesteuert, wo im Dateisystem die jeweiligen Datenbanken gespeichert werden. Dieser Konfigurationseintrag kann eine komplexe Syntax verarbeiten, mit der mehrere Verzeichnisse als Speicherorte angegeben werden.

Konfigurationssyntax:

```
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 ? ;
```

Beispiel:

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

Die Größenwerte sind optional. Sofern festgelegt, geben sie die maximale Gesamtgröße der hier gespeicherten Dateien an, bevor Datenbanken verschoben werden. Falls keine Größe angegeben wurde, wird die Datenbank nicht automatisch verschoben. Die Größe lässt sich jedoch auf andere Weise verwalten.

Die Verwendung von = oder == ist bedeutend. Datenbanken verhalten sich standardmäßig so, dass sie beim Starten des Core-Services automatisch Verzeichnisse erstellen. Allerdings lässt sich dieses Verhalten durch die Verwendung der Syntax == überschreiben. Wenn == verwendet wird, werden keinerlei Verzeichnisse durch den Service angelegt. Wenn die Verzeichnisse beim Start des Services nicht vorhanden sind, ist eine erfolgreiche Verarbeitung durch den Service nicht möglich. Dadurch wird dem Service eine gewisse Ausfallsicherheit gegenüber Dateisystemen verliehen, die beim Start des Hosts fehlen oder getrennt sind.

Wenn Sie die Größe eines bereits verwendeten Verzeichnisses ändern, wird die Änderung sofort wirksam, solange sie nach oben erfolgt. Wird die Größe nach unten geändert, wird sie ignoriert, wenn sie mehr als 10 Prozent kleiner als die derzeitige Größe ist. Dadurch wird ein versehentlicher Vertipper vermieden, durch den ein erheblicher Datenverlust entstehen könnte. Wenn die Paketdatenbank beispielsweise für `12 TB` konfiguriert wurde und jemand versehentlich `12 GB` eingegeben hat, würden in der Datenbank mehr als 11 TB an Daten gelöscht werden, um sie auf nur 12 GB zu schrumpfen. Stattdessen ignoriert die Datenbank die Vorgabe `12 GB` und protokolliert eine Warnmeldung, sodass der Fehler schnell ermittelt werden kann. Wenn die angegebene Größe richtig ist und sich um mehr als 10 % von der aktuellen Größe unterscheidet, wird sie selbstverständlich erst bei einem Neustart des Services wirksam. Bei einem Service-Neustart interpretiert das System die Größenangabe als richtig und nimmt eine Anpassung der Datenbankgröße vor, indem die ältesten Daten ausgelagert werden, um die neue Größe zu erreichen. Wenn Sie beabsichtigen, die Größe ohne einen Neustart des Systems um mehr als 10 Prozent zu verringern, müssen Sie die Größe in mehreren Vorgängen anpassen und sie jedes Mal um weniger als 10 Prozent verringern. Beobachten Sie die Serviceprotokolle, um einen Überblick zu behalten, wann die Datenbank die neue Größe erreicht hat, da die Gesamtgröße der Datenbank erst nach Schließen der letzten geschriebenen Datei angepasst wird.

Falls neue Verzeichnisse hinzugefügt oder gelöscht wurden (durch Semikolon getrennt), werden diese Änderungen erst nach einem Neustart des Services wirksam.

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

Diese Einstellungen sind optional und werden für die Warm-Tier-Speicherfunktion auf einem Archiver verwendet. Standardmäßig sind sie leer und nicht in Gebrauch. Wenn sie konfiguriert sind, weisen Sie dasselbe Format und Verhalten auf wie `packet.dir`, `meta.dir` und `session.dir` (siehe _ `packet.dir`, `meta.dir` und `session.dir` _ weiter oben). Sofern konfiguriert, wird die älteste Datei im Hot Tier in den Warm Tier verschoben, wenn der Platz im Hot Tier nicht mehr ausreicht.

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

Diese Einstellungen sind optional und finden Anwendung, wenn Dateien aus einem Hot- bzw. Warm-Tier-Speichersystem in ein angegebenes Cold-Tierverzeichnis verschoben werden. Bei dieser Einstellung handelt es sich um nichts anderes als ein Verzeichnis; es gibt keine Größenangaben. Der angegebene Pfadname weist jedoch einige spezielle Formatvorgaben auf, die Sie für die Benennung des Verzeichnisses mit dem Datum der darin enthaltenen Daten verwenden können.

```
%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
```

Beispiel für eine Einstellung:

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

Wenn bei der oben genannten Einstellung eine Datenbankprotokolldatei in den Cold-Speicher verschoben werden soll, die am `2014-03-02 15:00:00` erstellt wurde, würde sie in das folgende Verzeichnis im Cold Tier verschoben werden:

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

Die letzte Ziffer `01` bedarf einer gewissen Erläuterung. Die Spezifizierung `%8r` unterteilt die Stunden eines Tages in 24/8 = 3 Teile. Die ersten acht Stunden des Tages sind Block `00` , also 0 Uhr bis 8 Uhr. Die nächsten acht Stunden von 8 Uhr bis 16 Uhr entsprechen Block `01` . Da die Daten, die in den Cold-Speicher verschoben werden sollen, um 15 Uhr erstellt wurden, zählen sie zu Block `01` . Die Formatspezifizierung `%r` eignet sich für die Sicherung von Dateien mit einer Detailgenauigkeit etwa zwischen einem Tag `%d` und einer Stunde `%h` . Die Erstellung des Cold-Speichers erfolgt nach Bedarf und ergibt sich aus den verschobenen Daten bei Verwendung der Formatspezifizierungen.

Die Möglichkeit, dem Datenpfad ein Datum hinzuzufügen, ist für Sicherungs- und Wiederherstellungszwecke hilfreich. Dabei werden die Daten mit einem Tag im Pfad versehen.

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

Hiermit wird die Größe der mit jeder Datenbank erstellten Dateien gesteuert. Normalerweise ist es nicht notwendig, diese Werte zu ändern, da die Standardwerte im Allgemeinen gut funktionieren. Diese Einstellung ist für nachfolgende Dateien sofort wirksam.

packet.files,meta.files,session.files

Mit dieser Einstellung wird die Anzahl der von der Datenbank offen gehaltenen Dateien gesteuert. Dieser Wert lässt sich erhöhen, um die Performance zu verbessern, wobei das Betriebssystem über eine Obergrenze für gleichzeitig geöffnete Dateien des Systems verfügt. Wird diese Grenze überschritten, gibt es eine Fehlermeldung, und der Service funktioniert nicht. Diese Einstellung wird sofort wirksam.

In NetWitness Suite 10.6 und höher lautet der Standardwert für packet.files, meta.files und session.files `auto` und der Service managt die Anzahl der offenen Dateien auf Grundlage der folgenden Kriterien:

1. Anzahl der Sammlungen
2. Menge des Systemspeichers

Bei Festlegung auf `auto` ist die Zahl dynamisch und Sie können die Änderungen in den Protokollen sehen. Für NetWitness Suite 10.6 empfiehlt RSA, diesen Wert auf `auto` festzulegen und ihn nicht in eine bestimmte Zahl zu ändern.

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

Mit dieser Einstellung wird eine Sicherheitsbeschränkung für den Mindest-Speicherplatz der jeweils von packet.dir, meta.dir und session.dir angegebenen Verzeichnisse festgelegt. Damit wird verhindert, dass dem Service der Speicherplatz ausgeht, wenn andere Programme den Platz nutzen, der normalerweise jeder dieser Datenbanken zugewiesen sein sollte. Diese Einstellung wird sofort wirksam.

packet.index.fidelity,meta.index.fidelity

Mit dieser Einstellung wird gesteuert, in welchen Zeitabständen die Paket-ID- und Meta-ID-Speicherorte indiziert werden sollen. Die Einstellung kann erhöht werden, um den für jedes Paket bzw. jede nwindex-Metadatei benötigten Speicherplatz zu verringern, wodurch sich jedoch die Geschwindigkeit verringert, mit der einzelne Pakete oder Metaelemente gefunden werden können. Diese Einstellung wird sofort wirksam.

Für die Sitzungsdatenbank gibt es keine Genauigkeitseinstellungen, da sie keine Indexdateien erstellt.

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

Mit dieser Einstellung wird gesteuert, ob die Datenbank nach dem Schreiben einer Datei eine Synchronisierung des Dateisystems erzwingt. Der Standardwert ist `sync` . Das bedeutet, dass es bei einer geschlossenen Datei zu einer erheblichen Verzögerung kommt, wenn die Daten in einen nicht flüchtigen Speicher geschrieben werden. Es kann erforderlich sein, diesen auf `normal` zu setzen, um höhere kontinuierlichere Schreibgeschwindigkeiten zu erreichen, insbesondere auf einem Decoder. Diese Einstellung wird ab der nächsten erstellten Datei wirksam. Daher ist davon auszugehen, dass nach dem Setzen des Werts auf `normal` mindestens eine weitere Synchronisierung erfolgt.

Sollte es zu Paketverlusten kommen und packet.integrity.flush weist die Einstellung `sync` auf, setzen Sie es auf `normal` und überwachen Sie den Vorgang. Belassen Sie die Einstellungen für die Sitzung und Meta-Leerung bei `sync` . Sollte das Problem mit den Paketverlusten weiter bestehen, setzen Sie alle drei auf `normal` und überwachen Sie den Vorgang.

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

Die Blockgröße zeigt an, welche Datenmenge jeweils in eine Datenbankdatei geschrieben wird. Durch größere Blockgrößen lassen sich höhere Durchsatz- und Komprimierungsraten erreichen, zudem wird die Geschwindigkeit verbessert, mit der Elemente nacheinander aus der Datenbank abgerufen werden können. Größere Blockgrößen können sich jedoch nachteilig auf Random-Lesegeschwindigkeiten für komprimierte Paket- und Metaelemente auswirken. Diese Einstellung wird sofort wirksam.

packet.compression,meta.compression

Mit diesen Parametern wird gesteuert, ob die Datenbanken Daten komprimieren. Durch Komprimierung wird der von den Datenbanken benötigte Speicherplatz verringert, was sich jedoch sehr negativ sowohl auf die Geschwindigkeit auswirken kann, mit der Elemente in die Datenbank geschrieben werden, als auch auf die Geschwindigkeit, mit der Elemente aus der Datenbank abgerufen werden. Die Änderungen sind sofort ab Erstellung der nächsten Datei wirksam.

Ab NetWitness Suite 10.4 lauten die gültigen Werte für diesen Parameter `gzip` , `bzip2` , `lzma` oder `none` . `gzip` ist der bevorzugte Algorithmus, wenn die Komprimierung zum Einsatz kommt, da hier eine ausgewogene Balance zwischen Performance und gespartem Speicherplatz erzielt wird. Mit `bzip2` und `lzma` wird zwar jeweils mehr Speicherplatz gespart, was sich jedoch ziemlich stark auf die Geschwindigkeit auswirkt. Diese beiden Parameter sollten daher nur für geringe Verarbeitungsgeschwindigkeiten verwendet werden oder dann, wenn es insbesondere auf den Speicherplatz ankommt.

packet.compression.level,meta.compression.level

Mit diesen Einstellungen können Sie die Komprimierungsalgorithmen genauer einstellen. Bei deaktivierter Komprimierung haben sie keinerlei Auswirkungen. Die gültigen Werte liegen zwischen 0 und 9\. Der Standardwert lautet 0, bei dem die optimale Einstellung für Geschwindigkeit und Komprimierung von der Software ausgewählt wird. Die Werte von 1 bis 9 entsprechen einer Gleitskala zwischen Performance (1) und Komprimierung (9). Beim Wert 9 erhalten Sie für einen gegebenen Algorithmus die beste Komprimierung, jedoch die schlechteste Performance. Die beste Einstellung liegt meist im mittleren Bereich, welcher bei der Einstellung 0 ausgewählt wird.

hash.algorithm

Mit dieser Einstellung wird das Hashen der Datenbankdateien festgelegt. Der Standardwert ist `none` , es erfolgt also kein Hashing. Gültige Werte sind `none` , `sha256` , `sha1` und `md5` . Datenbankdateien können gehashed werden, um nachzuweisen, dass sie nach dem Schließen nicht manipuliert worden sind. Hashing-Vorgänge sind zeitaufwändig und beeinträchtigen, wenn sie aktiviert sind, die Verarbeitungs-Performance. Diese Änderung tritt sofort in Kraft.

hash.databases

Mit dieser Einstellung wird festgelegt, welche Datenbanken gehashed werden. Gültige Werte sind `session` , `meta` und `packet` . Beim Hashing von mehreren Datenbanken werden die Werte durch Kommas getrennt. Diese Änderung tritt sofort in Kraft.

hash.dir

Diese Einstellung ist normalerweise leer, d. h. die Hash-Datei wird in dasselbe Verzeichnis geschrieben wie die Datenbankdatei, die gehashed wurde. Bei Festlegung dieser Einstellung wird die Hash-Datei stattdessen in das angegebene Verzeichnis geschrieben. Das könnte eine Art Speicher zur Einmalverwendund sein, um gegen Hash-Manipulation gewappnet zu sein.

Bei Hash-Dateien handelt es sich um kleine XML-Dateien, die neben Metadaten den hex-kodierten Hash der Datei, die gehashed wurde, enthalten.

Index--Konfigurations-Nodes

In diesem Thema werden Indexkonfigurations-Nodes beschrieben. Die folgenden Indexkonfigurations-Nodes gehören zu den erweiterten Datenbank-Konfigurationselementen der NetWitness-Core-Datenbank, die nur selten geändert werden.

index.dir

Die Einstellung `index.dir` steuert, wo die vom Index verwendeten Dateien gespeichert werden. Diese Einstellung unterstützt die gleiche Syntax wie die Einstellungen `packet.dir` , `meta.dir` und `session.dir` .

index.dir.warm

Der Warm-Tier-Speicher für Index-Slices. Diese Einstellung unterstützt die gleiche Syntax wie `packet.dir.warm` , `meta.dir.warm` und `essionsession.dir.warm` .

index.dir.cold

Der Cold-Tier-Speicher für Index-Slices. Diese Einstellung unterstützt die gleiche Syntax wie `packet.dir.cold` , `meta.dir.cold` und `session.dir.cold` .

index.slices.open

Mit dieser Einstellung wird die Anzahl der vom Index offen gehaltenen Indexslices gesteuert. Indexslices werden automatisch geöffnet, wenn dies für Abfragen erforderlich ist. Wenn die Abfragen abgeschlossen sind, kann die Indexengine die Slices offen halten, damit weitere Abfragen schneller ausgeführt werden. Die zuletzt erstellen Slices werden offen gehalten, da sie höchst wahrscheinlich von Abfragen verwendet werden.

Wenn der Index für Abfragen im Index erst Slices öffnen muss, werden die Abfragen langsamer ausgeführt als bei bereits geöffneten Slices. Daher sollte dieser Parameter so eingestellt sein, dass die meisten Abfragen im Index an geöffneten Slices durchgeführt werden. Alle offenen Indexslices benötigen jedoch Ressourcen, wie z. B. Dateiverarbeitungen und Arbeitsspeicher. Wenn zu viele Indexslices offen sind, kann die Gesamtperformance des Services beeinträchtigt sein.

Sie sollten diesen Parameter so einstellen, dass die offenen Indexslices die meisten Datumsbereiche abdecken, die von den meisten Abfragen verwendet werden. Wenn die meisten Abfragen z. B. die letzten beiden Wochen betreffen und alle 8 Stunden Indexslices erstellt werden, wurden 14 Tage x 3 Slices pro Tag, also 42 Slices zu den letzten beiden Wochen erstellt. Somit können Sie index.slices.open auf 42 setzen, sodass nur die Slices offen gehalten werden, die sehr wahrscheinlich verwendet werden.

Wenn dieser Parameter auf 0 gesetzt wird, werden bis zum nächsten Speichern des Index alle Slices offen gehalten. In diesem Szenario wird die Anzahl der im Prozess offenen Slices nur durch die Anzahl der Slices im Index begrenzt.

`page.compression`

Veraltet. Versionen des NetWitness-Core-Index von 9.8 bis 10.2 unterstützten zwei verschiedene Indexkomprimierungsalgorithmen. Mit dieser Einstellung können Sie einen von ihnen auswählen. Ab 10.3 ist der einzige empfohlene Wert die Standardeinstellung `huffhybrid`.

`save.session.count`

Mit dieser Einstellung wird kontrolliert, wie häufig der Index automatisch gespeichert wird, wenn neue Sitzungen eingefügt werden. Wenn der Wert für save.session.count größer 0 ist, wird der Index jedes Mal automatisch gespeichert, wenn dem Index mehr als in `save.session.count` angegebene Sitzungen hinzugefügt werden. Wenn `save.session.count` auf 0 gesetzt wird, wird diese Funktion deaktiviert und der Index wird nicht automatisch gespeichert, wenn ihm neue Sitzungen hinzugefügt werden.

Mit dem Parameter `save.session.count` kann ein automatisches Speicherungsmuster implementiert werden, das auf der Menge der Daten basiert, die zum Index hinzukommen. Dies ist nützlich, da ein nur leicht belastetes System weniger häufig Sicherungspunkte erzeugen muss.

Weitere Informationen zur Speicherung des Index finden Sie im Abschnitt Optimierungstechniken in diesem Handbuch.

`reindex.enable`

Diese Einstellung steuert den Betrieb des Hintergrund-Reindexer .

SDK-Konfigurations-Nodes

In diesem Thema werden die SDK-Konfigurations-Nodes beschrieben, die die Datenbank betreffen. Es gibt einige zusätzliche Konfigurationselemente in jedem Core-Service, die sich zwar auf die Datenbank auswirken, aber keine tatsächlichen Auswirkungen darauf haben, wie die Datenbank Daten speichert oder abruft. Diese Einstellungen befinden sich in dem Ordner `/sdk/config`.

max.concurrent.queries

Diese Einstellung legt fest, wie viele Abfragevorgänge gleichzeitig auf der Datenbank erlaubt sind. Das Erlauben mehrerer gleichzeitiger Abfragevorgänge kann die allgemeine Reaktionsgeschwindigkeit für mehr Benutzer verbessern, aber wenn die Abfragelast des Core-Services sehr I/O-gebunden ist, kann ein hoher Wert für „max.concurrent.queries“ schädliche Auswirkungen haben. Der empfohlene Wert ist nahe der Anzahl von Cores auf dem System, einschließlich Hyper-Threading. So sollte der Wert für eine Appliance mit 16 Cores etwa bei 32 liegen. Ziehen Sie einige ab für Aggregationsthreads und allgemeine Systemreaktionsthreads. Ziehen Sie einige weitere ab, wenn dies ein hybrides System ist (wenn zum Beispiel sowohl ein Decoder als auch ein Concentrator auf derselben Appliance laufen). Es gibt keine magische Anzahl, aber jede zwischen 16 und 32 sollte gut funktionieren.

max.pending.queries

Diese Einstellung kontrolliert die Größe des Backlogs für die Abfrage-Engine der Datenbank. Größere Werte erlauben, dass die Datenbank mehr Vorgänge zur Ausführung in die Warteschlange stellen kann. Eine Abfrage, die in die Warteschlange gestellt wurde, macht keinen Fortschritt bei ihrer Ausführung, daher kann es nützlicher sein, das System Fehler machen zu lassen, wenn die Warteschlange voll ist, anstatt zu erlauben, dass die Warteschlange sehr groß wird. Allerdings kommt es bei einem System, das in erster Linie Batchvorgänge wie Berichte durchführt, möglicherweise nicht zu negativen Auswirkungen durch eine große Warteschlange.

cache.window.minutes

Diese Einstellung steuert eine Funktion der Abfrage-Engine, die die Reaktionsgeschwindigkeit der Abfrage verbessern soll, wenn gleichzeitig viele Benutzer zugreifen. Weitere Informationen über das Cachefenster finden Sie unter Optimierungstechniken .

max.where.clause.cache

Der Where-Klausel-Cache legt fest, wie viel Arbeitsspeicher von Abfragevorgängen konsumiert werden kann, die eine große temporäre Datenmenge produzieren müssen, um das Sortieren oder Zählen auszuwerten. Wenn die Größe des Where-Klausel-Caches nicht ausreicht, funktioniert die Abfrage zwar noch, aber sie ist viel langsamer. Wenn der Where-Klausel-Cache zu groß ist, ist es möglich, dass Abfragen so viel Arbeitsspeicher allokieren, dass der Service in den Swap-Modus gezwungen würde oder zu wenig Arbeitsspeicher hätte. Daher sollte dieser Wert, multipliziert mit dem Wert max.concurrent.queries, immer kleiner sein als die Größe des physischen RAM. Diese Einstellung versteht Größen in Form einer Zahl, gefolgt von einer Einheit, zum Beispiel `1.5 GB` .

max.unique.values

Die maximalen eindeutigen Werte begrenzen, wie viel Arbeitsspeicher von der Funktion „SDK-Werte“ beansprucht werden kann. SDK-Werte erzeugt eine sortierte Liste eindeutiger Werte. Damit genauere Ergebnisse erzeugt werden können, müssen möglicherweise eine große Anzahl eindeutiger Werte aus vielen Slices zusammengeführt werden. Dieser zusammengeführte Satz an Werten muss im Arbeitsspeicher eingeschränkt werden. Dieser Parameter existiert, um zu begrenzen, wie viel Arbeitsspeicher der zusammengeführte Wertesatz belegen kann. Der Standardwert wird begrenzt die Nutzung des Arbeitsspeichers auf ca. 1/10 der gesamten RAM.

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

Diese Einstellungen sind in NetWitness Suite 10.4 und älteren Versionen verfügbar.

In NetWitness Suite 10.4 und älteren Versionen unterstützt die Core-Datenbank drei Abfrageprioritätsstufen. Jeder Benutzer wird einem der Prioritätslevel zugewiesen. Daher gibt es bis zu drei Gruppen von Benutzern, die für die Zwecke des Performance-Tuning definiert werden können. Diese Einstellungen legen fest, wie lange es jeder Benutzerebene erlaubt ist, die Abfragen auszuführen. Beispielsweise kann für Benutzer mit geringeren Berechtigungen ein niedrigerer Wert festgelegt werden, sodass sie nicht durch lange dauernde Abfragen die gesamten Ressourcen des Core-Services verbrauchen können.

query.timeout

Diese Einstellung ist in NetWitness Suite 10.5 und neueren Versionen verfügbar.

Abfrageebenen wurden in NetWitness Suite 10.5 und neueren Versionen durch Abfrage-Timeouts pro Benutzerkonto ersetzt. Für vertrauenswürdige Verbindungen werden diese Timeouts auf dem NetWitness Suite-Server konfiguriert. Für Konten auf Core-Services gibt es einen neuen Konfigurations-Node mit der Bezeichnung `query.timeout` unter jedem Konto. Dieser gibt die maximale Zeit in Minuten an, die jede Abfrage ausgeführt werden kann. Das Festlegen der Werte auf null bedeutet, dass kein Abfragetimeout durch den Core-Service durchgesetzt wird.

max.where.clause.sessions

Diese Einstellung ist in NetWitness Suite 10.5 und neueren Versionen verfügbar.

Diese Einstellung begrenzt, wie viele Sitzungen durch eine einzige Abfrage gescannt werden können. Beispiel: Wenn ein Benutzer alle Metadaten einer Datenbank auswählt, beendet die Datenbank die Verarbeitung der Ergebnisse, sobald die Anzahl der Sitzungslesevorgänge für die Abfrage diesen Konfigurationswert erreicht. Der Wert 0 deaktiviert diese Begrenzung.

Die Anzahl der für die vollständige Verarbeitung einer Abfrage erforderlichen Sitzungen entspricht der Anzahl der Sitzungen, die mit der WHERE-Klausel dieser Abfrage übereinstimmen, vorausgesetzt für alle Begriffe in der Where-Klausel ist ein passender Index vorhanden. Wenn die Where-Klausel nicht indexierte Begriffe enthält, muss die Datenbank mehr Sitzungen und Metadaten lesen und die Begrenzung wird schneller erreicht.

max.query.groups

Diese Einstellung ist in NetWitness Suite 10.5 und neueren Versionen verfügbar.

Diese Einstellung begrenzt die Anzahl der eindeutigen Gruppen, die in einer einzigen Abfrage gesammelt werden. Beispiel: Wenn eine Abfrage eine Gruppierungsklausel mit mehreren Metadaten mit hohen eindeutigen Wertanzahlen enthält, kann der Speicherbedarf für diese Abfrage leicht den auf dem Server verfügbaren Arbeitsspeicher überschreiten. Diese Begrenzung gibt es also, um das Auftreten von „Out of Memory“-Bedingungen zu verhindern.

Das Festlegen des Werts 0 deaktiviert diese Begrenzung.

packet.read.throttle

Dies ist eine reine Decoder-Einstellung, die Auswirkungen auf den Zugriff auf die Pakete in der Datenbank hat. Wenn packet.read.throttle auf einen Wert größer 0 festgelegt wird, versucht der Decoder, das Lesen von Paketen zu drosseln, wenn ein Paketkonflikt in der Paketdatenbank erkannt wird. Hohe Zahlen entsprechen einer stärkeren Drosselung. Die Änderungen werden sofort wirksam.

cache.dir,cache.size

Alle NetWitness Suite-Core-Services verfügen über einen kleinen Dateicache mit Rohinhalten, die von dem Gerät extrahiert wurden. Diese Parameter steuern den Speicherort (cache.dir) und die Größe (cache.size) dieses Caches.

parallel.values

Diese Einstellung ist in NetWitness Suite 10.5 und neueren Versionen verfügbar.

Diese Einstellung ermöglicht das parallele Ausführen von SDK-Wertoperationen. Wird hier 0 festgelegt, wird die parallele Ausführung deaktiviert. Wenn ein Wert größer 0 festgelegt wird, steht dieser für die Anzahl an Threads, die erstellt werden, wenn jede SDK-Wertoperation ausgeführt wird. Der Maximalwert ist gleich der Anzahl der beim Start des Prozesses verfügbaren logischen CPUs.

Das Festlegen eines höheren Wertes für parallel.values ist bei einer geringen Anzahl an gleichzeitigen Benutzer hilfreich, da dadurch eine schnellere Ausführung komplexerer Ermittlungen ermöglicht wird. Bei vielen gleichzeitigen Benutzern ist ein niedrigerer Wert besser, da viele unabhängige SDK-Wertoperationen gleichzeitig ausgeführt werden.

parallel.query

Diese Einstellung ist in NetWitness Suite 10.5 und neueren Versionen verfügbar.

Diese Konfiguration ähnelt der Einstellung parallel.values insofern, als dass der Maximalwert gleich der Anzahl an logischen CPUs ist. Beim Festlegen von parallel.query auf einen bestimmten Wert sollte die Anzahl der gleichzeitigen Benutzer berücksichtigt werden, um die CPU-Auslastung zu maximieren, ohne dabei ständig die verfügbaren Ressourcen zu überschreiten.

Das Festlegen eines höheren Wertes für parallel.query ist bei einer geringen Anzahl an gleichzeitigen Benutzer und Abfragen hilfreich, da dadurch eine schnellere Ausführung komplexerer Abfragen ermöglicht wird. Bei vielen gleichzeitigen Benutzern und Abfragen ist ein niedrigerer Wert besser, da viele unabhängige SDK-Abfrageoperationen gleichzeitig ausgeführt werden.

Abfrageoperationen werden durch die Lesegeschwindigkeit der Metadatenbank beschränkt. Durch das Festlegen von parallel.query auf einen Wert über 4 werden daher vermutlich keine signifikant besseren Ergebnisse als beim Standardwert von 0 erzielt werden. Die optimale Anzahl für parallel.query ist abhängig von dem installierten Speicher. Testen Sie verschiedene Werte für parallel.query, um das optimale Ergebnis für Ihr Speichersystem zu ermitteln.

Per-User Konfigurations-Node

In diesem Thema werden die Per-User Konfigurations-Nodes beschrieben. Über bestimmte Einstellungen wird beeinflusst, welche Aktionen Benutzer mit der Datenbank durchführen dürfen. Diese Einstellungen werden in der Konfigurationsstruktur unter `/users/accounts/<username>/config` gespeichert. Dabei gilt Folgendes: `<username>` ist der Name des Benutzers, für den die Einstellungen gelten.

query.prefix

Über ein Abfragepräfix wird ein Filter auf jede Abfrageoperation angewendet, die vom Benutzer durchgeführt wird. Dafür werden die `query.prefix` -Werte mithilfe des logischen Und-Operators (&&) an die WHERE-Klauseln jeder Abfrage angehängt. Weitere Informationen zu WHERE-Klauseln finden Sie unter Abfragen .

query.level

Diese Einstellung ist in NetWitness Suite 10.4 und früheren Versionen verfügbar.

Über die query.level-Einstellung wird die Abfragenebene zugewiesen, die von den Benutzern für jede von ihnen durchgeführte Abfrage verwenden können. Das hat Einfluss darauf, ob die Abfragen durch query.level.1.minutes, query.level.2.minutes oder query.level.3.minutes begrenzt sind.

query.timeout

Diese Einstellung ist in NetWitness Suite 10.5 und neueren Versionen verfügbar.

Über die query.timeout-Einstellung wird die maximal zulässige Dauer in Minuten festgelegt, die ein Benutzer eine Abfrage ausführen kann. Für vertrauenswürdige Verbindungen werden diese Timeouts auf dem NetWitness Suite-Server konfiguriert. Für Konten in Core-Services wird diese Einstellung in der Konfigurationsstruktur unter `/users/accounts/<username>/config` gespeichert. Dabei gilt Folgendes: `<username>` ist der Name des Benutzers, für den die Einstellungen gelten. Wenn dieser Wert auf Null eingestellt ist, wird das Abfragetimeout durch den Core-Service nicht erzwungen.

session.threshold

Über die session.threshold-Einstellung wird ein maximaler Schwellenwert für den Benutzer festgelegt. Wenn diese Einstellung festgelegt ist, wird der Schwellenwert auf alle Wertaufrufe angewendet, die von dem Benutzer durchgeführt werden. In dieser Anleitung finden Sie eine detaillierte Beschreibung von Wertaufrufen und Schwellenwerten.

Scheduler

In diesem Thema wird eine kurze Einführung in die Funktionen des Scheduler bereitgestellt und erklärt, wie Befehle geplant werden. Alle NetWitness Core-Services verfügen über einen integrierten Scheduler, der unter `/sys/config/scheduler` zu finden ist. Um den Scheduler zu verwenden, geben Sie mithilfe einer der folgenden zwei Meldungen den Befehl ein, der regelmäßig ausgeführt werden soll:

`/sys/config/scheduler addInter` – Geben Sie einen Befehl ein, der im angegebenen Intervall ausgeführt werden soll (alle n Stunden, Minuten oder Sekunden).

oder

`/sys/config/scheduler addMil` – Geben Sie einen Befehl ein, der zu einer bestimmten Zeit oder an einem bestimmten Wochentag ausgeführt werden soll.

Beispiel

Angenommenes Anwendungsbeispiel: Alle Paketdaten, die älter als sieben Tage sind, sollen gelöscht werden. Da Sie die packet.dir-Einstellung nicht für einen Daten-Rollout basierend auf einem Zeitintervall konfigurieren können, müssen Sie den Befehl `/database timeRoll` für eine regelmäßige Ausführung konfigurieren. Erstellen Sie für dieses Beispiel den Befehl „timeRoll“, der alle 20 Minuten ausgeführt wird:

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

Dieser Befehl fügt dem Node `/database` eine geplante Aufgabe hinzu, die alle 20 Minuten ausgeführt wird, und sortiert alle Paketdaten aus, die älter als sieben Tage sind. Der Parameter `params` wird verwendet, um alle Parameter an den festgelegten Befehl weiterzugeben (in diesem Fall `timeRoll`). Beachten Sie, dass die eingebetteten Parameter (`type` und `days`) in Anführungszeichen stehen, sodass sie nicht als Parameter interpretiert werden, die an den äußeren Befehl `addIter` weitergegeben werden müssen. Wenn die Parameter innerhalb von `params` in Anführungszeichen stehen müssen, müssen sie mit einem rückwärtigen Schrägstrich abgetrennt werden. Sie können den Befehl mit integrierten Anführungszeichen neu schreiben, ohne diesen dadurch zu verändern:

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

Der Befehl erfüllt die gleiche Funktion wie das Original, zeigt jedoch, wie komplizierte Parameterweitergaben vermieden werden können. Weitere hilfreiche Befehle für den Scheduler sind:

`/sys/config/scheduler print` – Druckt alle geplanten Befehle (Sie können diese auch mit einem `ls` auf dem Scheduler-Node anzeigen.)

`/sys/config/scheduler delSched` – Löschen Sie einen geplanten Befehl, indem die im Befehl `print` (oder `ls`) gezeigte Kennung übergeben.

Dies ist eine kurze Einführung in die Funktionen des Scheduler. Senden Sie für weitere Informationen zu Befehlsparametern die Nachricht `help` an den Scheduler-Node und geben Sie den Befehlsnamen über den Parameter `msg` ein. Weitere Informationen finden Sie im Thema „Ansicht „Durchsuchen“ für einen Service“ im *Leitfaden für die ersten Schritte mit Hosts und Services* .

Rollover

In diesem Thema werden die zwei Rollover-Mechanismen beschrieben. Die Datenbank operiert als FIFO-Warteschlange (First In, First Out). Neue Daten werden immer an die Datenbank angehängt und die ältesten Daten werden bei Bedarf automatisch entfernt. Daten in der Mitte der Datenbank sind unveränderbar und können nicht modifiziert werden.

Es gibt zwei Mechanismen für das Rollover: synchron und asynchron.

Synchrones Rollover

Das synchrone Rollover bezieht sich auf Rollover-Einstellungen, die als Reaktion auf einen Schreibvorgang in der Datenbank angewendet werden. Das bedeutet, dass Daten direkt deshalb aus der Datenbank entfernt werden, weil neue Daten geschrieben werden müssen. Das synchrone Rollover wird konfiguriert, indem Sie Größenwerte bei der Konfiguration für `packet.dir` , `meta.dir` , `session.dir` und `index.dir` festlegen.

Das synchrone Rollover in den Paket-, Meta- und Sitzungsdatenbanken kann innerhalb jedes beliebigen Schreibvorgangs auftreten. Das synchrone Rollover beim Index findet statt, wenn der Index gespeichert wird.

Asynchrones Rollover

Beim asynchronen Rollover wird die Datenbankdatei entfernt; er tritt auf, wenn ein expliziter Rollover-Befehl an die Datenbank ergeht. In der Regel wird diese Art des Rollover mithilfe der integrierten Planungsfunktion des Core-Service regelmäßig ausgeführt. Der Benutzer kann ihn auch ausdrücklich anfordern.

Der Befehl für das asynchrone Rollover ist die Meldung `sizeRoll` in den Nodes `/index` und `/database` der Konfigurationsstruktur. Die Meldung auf dem Node `/database` bewirkt ein Größen-Rollover ausschließlich an Paket-, Meta- und Sitzungsdatenbanken, während die Meldung auf dem Node `/index` simultane Rollover sowohl am Index als auch an Paket-, Meta- und Sitzungsdatenbanken durchführen kann.

Der Befehl `sizeRoll` hat die folgende Parametersyntax:

```
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 ? ;
```

Der Parameter `type` steuert die Datenbanken dahingehend, dass die Entfernung der ältesten Daten basierend auf der Gesamtgröße oder dem verbleibenden Speicherplatz in Erwägung gezogen wird. Wenn „type“ beim `/index` sizeRoll nicht angegeben ist, wird für die Rollover-Vorgänge nur der Index berücksichtigt.

Der Parameter `maxSize` legt eine aktuelle Maximalgröße für die Datenbank oder den Index fest. Wenn die Datenbank größer als dieser Wert ist, werden die ältesten Daten zuerst gelöscht (oder abhängig von der Konfiguration in den Warm oder Cold Tier verschoben), bis die Gesamtgröße unter `maxSize` sinkt. Der `sizeRoll` -Vorgang stellt fest, welche Daten basierend auf Sitzungs-IDs die ältesten aus allen Datenbanken und dem Index sind. Sitzungen oder Indexeinträge mit den niedrigsten Sitzungs-IDs werden zuerst gelöscht und möglicherweise werden auch Meta- und Paketdatenbanken entfernt, die durch die Entfernung der Einträge aus der Sitzungsdatenbank verwaist sind. Für die Indexdaten wird ein Rollout durchgeführt, wenn die Sitzungen, auf die sie sich beziehen, entfernt werden.

Der Parameter `maxSizeWarm` legt eine aktuelle Maximalgröße auf dem *Warm* Tier fest, verhält sich jedoch ansonsten genau wie der Parameter `maxSize` . Wenn für die Daten ein Rollout aus dem Warm Tier durchgeführt wird, werden sie in den Cold Tier verschoben (sofern konfiguriert) oder gelöscht.

Der Parameter `maxPercent` legt einen kombinierten maximalen Prozentsatz aller Volumes aller Datenbanken fest, die im Parameter `type` übergeben werden. Bei Überschreitung des Grenzwerts werden die ältesten Daten zuerst gelöscht, bis die Gesamtgröße unterhalb des maximalen Prozentsatzes für alle Volumes liegt.

Der Parameter `minFree` legt einen Mindestwert für den freien Speicherplatz fest, der in Volumes erlaubt ist, bevor die ältesten Daten gelöscht werden.

Jeder Aufruf an den sizeRoll-Vorgang sorgt für einen einzigen Durchgang durch die Datenbank zum Löschen von Dateien. Wenn der Vorgang abgeschlossen ist, erfüllt die aktuelle Größennutzung der Datenbank die von den Parametern `maxSize` , `maxPercent` oder `minFree` und dem optionalen `maxSizeWarm` angegebenen Kriterien. Deshalb kann dieser Vorgang regelmäßig eingeplant werden, um sicherzustellen, dass die Datenbank ohne Unterbrechung operieren kann.

Beispiel

Im folgenden Beispiel wird ein typischer sizeRoll-Planungseintrag für einen Archiver gezeigt:

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

Dieser Planungseintrag gibt an, dass die Datenbank alle fünf Minuten sicherstellt, dass die Maximalgröße der Meta-, Sitzungs-, Paket- und Indexdaten 25 Terabyte auf dem Hot-Tier und 150 Terabyte auf dem Warm-Tier nicht überschreitet.

Abfragen

In diesem Thema wird die Datenbankabfragesyntax beschrieben. Es gibt drei Hauptmechanismen für das Durchführen von Abfragen in der Datenbank, die Aufrufe `query` , `values` und `msearch` im Ordner `/sdk` in jedem Core-Service.

Der Aufruf `query` gibt Metaelemente aus der Metadatenbank zurück, wobei der Index für ein schnelles Abrufen genutzt werden kann.

Der Aufruf `values` gibt Gruppen eindeutiger Metawerte zurück, die nach bestimmten Kriterien geordnet sind. Er ist so optimiert, dass er die eindeutigen Werte als Untergruppe zurückgibt, die von einer Aggregatfunktion, z. B. Anzahl, sortiert werden.

Der Aufruf `msearch` verwendet Suchbegriffe als Eingabe und gibt passende Sitzungen zurück, die mit den Suchbegriffen übereinstimmen. Er kann innerhalb von Indizes, Metadaten, Rohpaketdaten oder Rohprotokolldaten suchen.

`query` Syntax

Die Meldung `query` weist folgende Syntax auf:

```
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 ? ;
```

Die Parameter `id1` , `id2` und `size` bilden für die Rückgabe größerer Ergebnismengen aus der Datenbank einen Auslagerungsmechanismus. Sie werden vor allem von Entwicklern genutzt, die Anwendungen direkt für die NetWitness Core-Datenbank erstellen. Normalerweise werden zurückgegebene Ergebnisse vom ältesten zum neuesten Eintrag sortiert (höhere Meta-IDs sind stets aktueller). Um die Ergebnisse vom neuesten zum ältesten Eintrag zu sortieren, drehen Sie die IDs um, sodass `id1` größer ist als `id2` . Dies bringt eine leichte Verschlechterung der Performance mit sich, da die Where-Klausel zunächst vollständig ausgewertet werden muss, bevor die Verarbeitung in umgekehrter Reihenfolge beginnen kann.

Wenn keine Größe angegeben oder sie auf null gesetzt wurde, gibt das System alle Ergebnisse ohne Paging zurück. Für die RESTful-Schnittstelle wird die gesamte Antwort daher mit segmentierter Codierung zurückgegeben. Das systemeigene Protokoll gibt die Ergebnisse in mehreren Meldungen zurück.

Der Parameter `query` ist eine `query` -Befehlszeichenfolge mit eigener NetWitness-spezifischer Syntax:

```
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}
```

Mithilfe der `select` -Klausel können Sie entweder `*` festlegen, dass alle Metadaten aus allen Sitzungen zurückgegeben werden, die mit der Where-Klausel übereinstimmen, oder eine Reihe von Meta-Feldnamen und Aggregatfunktionen angeben, anhand derer bei jeder Sitzung eine Untergruppe der Metadaten ausgewählt wird.

Die Klausel `select` kann umbenannte Metaschlüsselnamen enthalten. Alle Felder, die im Ergebnissatz als Ergebnis eines umbenannten Schlüssels in der Klausel `select` angezeigt werden, werden mit den Namen des Metaschlüssels zurückgegeben, der dem in der Klausel `select` verwendeten Namen entspricht. Beispiel: Wenn der Schlüssel `port_src` zum Umbenennen von `tcp.srcport` verwendet wird, gibt eine Abfrage mit `select port_src` nur `port_src` -Felder zurück, selbst wenn die zugrunde liegenden Metadaten den Typ `tcp.srcport` aufwiesen.

Die Aggregatfunktionen wirken sich wie folgt auf den Ergebnissatz der Abfrage aus:

Funktion	Ergebnis
`sum`	Alle Metawerte zusammenaddieren; nur bei Ziffern möglich
`count`	Die Gesamtzahl von Metafeldern, die zurückgegeben worden wären
`min`	Der gesehene Mindestwert
`max`	Der gesehene Maximalwert
`avg`	Der Durchschnittswert für die Anzahl
`distinct`	Gibt eine Liste aller erkannten eindeutigen Werte zurück

`countdistinct`	Gibt die Anzahl der gesehenen eindeutigen Werte zurück. `countdistinct` ist gleich der Anzahl von Metawerten, die die Funktion „distinct“ zurückgegeben hätte.
`first`	Gibt den ersten gesehenen Wert zurück
`last`	Gibt den letzten gesehenen Wert zurück
`len`	Konvertiert alle Feldwerte in eine UInt32-Länge, statt den tatsächlichen Wert anzugeben. Diese Länge entspricht nicht der Länge der in der Metadatenbank gespeicherten Struktur, sondern der für die Speicherung des tatsächlichen Werts erforderlichen Anzahl von Byte. Für den Begriff „NetWitness“ wird beispielsweise die Länge 10 zurückgegeben. Für IPv4-Felder, z. B. `ip.src` , werden stets 4 Byte zurückgegeben.

`where` Klauseln

Bei der `where` -Klausel handelt es sich um einen spezifischen Filter, mit dem Sie unter Verwendung des Index Sitzungen aus der Sammlung auswählen können.

Syntax:

```
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" ;
```

Bei der Angabe von Regelkriterien muss der Teil der Klausel, der sich auf den `meta-value` bezieht, mit dem im `meta-key` angegebenen Metawert übereinstimmen. Wenn der Schlüssel beispielsweise `ip.src` ist, sollte der `meta-value` eine IPv4-Adresse sein.

Abfragen mit einem `meta-key` -Namen stimmen mit Metaelementen überein, die sowohl dem `meta-key` -Namen als auch den Namen aller für den Schlüssen angegebener „Umbenennungen“ entsprechen. Details zum Umbenennen von Schlüsseln finden Sie im Thema Indexanpassung .

Abfrageoperatoren

In der folgenden Tabelle werden die Funktionen der einzelnen Operatoren beschrieben:

Operator	Funktion
=	Entspricht Sitzungen, die den exakten Metawert enthalten. Bei Angabe eines Wertebereichs wird jeder der Werte als Übereinstimmung betrachtet.
!=	Entspricht allen Sitzungen, die mit ebendieser Klausel nicht übereinstimmen würden, wenn sie mit dem Operator = verfasst worden wäre.
<	Für numerische Werte: Entspricht Sitzungen, die Metadaten mit einem numerischen Wert enthalten, der kleiner ist als die rechte Seite. Wurde auf der rechten Seite ein Bereich angegeben, wird der erste darin genannte Wert herangezogen. Wenn mehrere Bereiche angegeben wurden, ist das Verhalten unbestimmt. Bei Metadaten in Textform erfolgt ein lexikografischer Vergleich.
<=	Dasselbe Verhalten wie bei < , wobei auch Sitzungen mit Metadaten, die dem Wert genau entsprechen, als Übereinstimmung gelten.
>	Ähnlich wie beim Operator < , wobei hier Sitzungen als Übereinstimmung gelten, deren numerischer Wert größer ist als die rechte Seite. Wurde die rechte Seite als Bereich angegeben, wird der letztgenannte Wert des Bereichs für den Vergleich berücksichtigt.
>=	Dasselbe Verhalten wie bei > , wobei auch Sitzungen mit Metadaten, die dem Wert genau entsprechen, als Übereinstimmung gelten.
`begins`	Entspricht Sitzungen, die Metadaten in Textform enthalten, die mit den gleichen Buchstaben wie die rechte Seite beginnen.

`ends`	Entspricht Sitzungen, die Metadaten in Textform enthalten, die mit den gleichen Buchstaben wie die rechte Seite enden.
`contains`	Entspricht Sitzungen, die Metadaten in Textform enthalten, die die auf der rechten Seite angegebene Teilzeichenkette enthalten.
`regex`	Entspricht Sitzungen, die Metadaten in Textform enthalten, die dem regulären Ausdruck (Regex) auf der rechten Seite entsprechen. Die Regex-Analyse erfolgt durch boost::regex.
`exists`	Entspricht Sitzungen, die beliebige Metawerte mit dem angegebenen Metaschlüssel enthalten.
`!exists`	Entspricht Sitzungen, die keine Metawerte enthalten, die dem angegebenen Metaschlüssel entsprechen.
`length`	Entspricht Sitzungen, die Metawerte in Textform mit einer bestimmten Länge enthalten. Der Ausdruck auf der rechten Seite muss eine nichtnegative Zahl sein.

Textwerte

Das System erwartet in Anführungszeichen stehende Textwerte. Wenn er nicht als Zeit (siehe unten) analysiert werden kann, wird ein in Anführungszeichen stehender Wert als Text interpretiert.

IP-Adressen

IP-Adressen lassen sich mithilfe von Standardtextausdrücken für IPv4- und IPv6-Adressen darstellen. Außerdem kann in der Abfrage die CIDR -Notation für den Ausdruck eines Adressbereichs verwendet werden. Bei Verwendung der CIDR-Notation wird diese auf den entsprechenden Wertebereich erweitert.

MAC-Adressen

Eine MAC-Adresse kann mithilfe der standardmäßigen MAC-Adressen-Notation angegeben werden: `aa:bb:cc:dd:ee:ff`

Ausdruck von Datum und Zeit

Datumsangaben in NetWitness Suite erfolgen mithilfe der Unix-Epochenzeit, die der Anzahl der seit dem 1.1.1970 (UTC) vergangenen Sekunden entspricht. In Abfragen können Sie die Zeit als Anzahl dieser Sekunden darstellen oder die Darstellung als Zeichenkette nutzen. Die Darstellung von Datum und Zeit als Zeichenkette lautet `"YYYY-mmm-DD HH:MM:SS"` . Der Monat wird als dreibuchstabige Abkürzung dargestellt. Sie können den Monat auch als zweistellige Zahl zwischen 01 und 12 angeben.

Zeitwerte müssen in Anführungszeichen gesetzt werden.

Es wird davon ausgegangen, dass in Abfragen enthaltene Zeitangaben im UTC-Format angegeben sind.

Relative Zeitpunkte

Mit relativen Zeitpunkten kann eine where-Klausel auf einen Wert mit einem bestimmten Versatz von den frühesten oder spätesten Zeitmetadaten in der Sammlung verweisen.

Ein Ausdruck für einen relativen Zeitpunkt weist die Syntax `rtp(boundary, duration)` auf.

Die Grenze ist entweder `earliest` oder `latest` .

Die Dauer ist ein Ausdruck von Stunden, Minuten oder Sekunden. Beispiel: `24h` , `60m` oder `60s` .

Relative Zeit Punkte können nur in SDK-Vorgängen mit einer Sammlung verwendet werden, aus der die Grenzen für die frühesten und spätesten Zeitmetawerte abgerufen werden können.

Relative Zeitpunkte können nur mit indizierten Metadatentypen verwendet werden. Standardmäßig sind `time` und `event.time` indizierte Metadatentypen.

Beispiele:

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


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

Werte für spezielle Bereiche

Bereiche werden üblicherweise mit der Syntax * `smallest` * - * `largest` * angegeben, es gibt allerdings einige spezielle Platzhalterwerte, die in Bereichsausdrücken verwendet werden können. Der Buchstabe `l` wird beispielsweise verwendet, um die untere Grenze aller Metawerte als Startwert des Bereichs zu verwenden, während `u` für die obere Grenze steht. Die Grenzen werden anhand des kleinsten bzw. größten Werts aus allen bereits in den Index eingegangenen Metawerten ermittelt.

Wenn Sie das Tag `l` oder `u` verwenden, sollte es nicht in Anführungszeichen stehen.

So würde der Ausdruck `time = "2014-may-20 11:57:00" - u` allen Zeitangaben ab dem 20. Mai 2014 11:57:00 bis zum neuesten in der Sammlung enthaltenen Datum entsprechen.

Beachten Sie, dass der Ausdruck eines Bereichs leicht mit einer Textzeichenkette verwechselt werden kann. Stellen Sie sicher, dass Textwerte, die - enthalten, in Anführungszeichen stehen und dass Bindestriche innerhalb von Ausdrücken eines Bereichs nicht innerhalb von Text in Anführungszeichen stehen.

`group by` -Klausel (seit 10.5)

Die Abfrage-API hat die Fähigkeit, aus den Ergebnissen eines Abfrageaufrufs Aggregatgruppen zu erzeugen. Dies geschieht, indem eine `group by` -Klausel auf die Abfrage angewendet wird. Wenn `group by` angegeben wird, wird der Ergebnissatz für die Abfrage in Gruppen unterteilt. Jede Ergebnisgruppe wird durch die in der group by-Klausel angegebenen Metawerte eindeutig identifiziert.

Ziehen Sie beispielsweise die Abfrage `select count(ip.dst)` in Betracht. Diese Abfrage gibt die Anzahl aller ip.dst-Metadaten in der Datenbank zurück. Bei Hinzufügen einer `group by` -Klausel wie `select count(ip.dst) group by ip.src` gibt die Abfrage jedoch eine Anzahl der ip.dst-Metadaten zurück, die für jede eindeutige ip.src gefunden wurde.

Ab NetWitness Suite Version 10.5 können Sie bis zu 6 Metafelder in einer `group by` -Klausel verwenden.

Die `group by` -Klausel hat mit der `values` -Abfrage einige Funktionen gemeinsam, aber sie bietet signifikant erweiterte Gruppen auf Kosten längerer Abfragezeiten. Zum Hervorbringen der Ergebnisse einer gruppierten Abfrage gehört das Lesen der Metadaten von der Metadatenbank für alle Sitzungen, die der `where` -Klausel entsprechen, während eine Values-Abfrage ihre Aggregate hervorbringen kann, indem sie einfach den Index liest.

Die Inhalte jeder Gruppe, die von der Abfrage zurückgegeben werden, werden durch die `select` -Klausel definiert. Die `select` -Klausel kann jede der ausgewählten Aggregatfunktionen oder Metafelder enthalten. Wenn mehrere Aggregate ausgewählt werden, wird das Ergebnis der Aggregatfunktion für jede Gruppe definiert. Wenn nicht aggregierte Felder ausgewählt werden, werden die Metafelder für jede Gruppe stapelweise zurückgegeben.

Der Ergebnissatz einer `group by` -Abfrage wird mit den folgenden Regeln kodiert:

1. Alle einer Gruppe zugeordneten Metadaten werden mit derselben Gruppennummer bereitgestellt.
2. Die ersten an die Gruppe zurückgegebenen Metadaten bestimmen den Gruppenschlüssel. Wenn beispielsweise die `group by` -Klausel `group by ip.src` angibt, ist das erste Metadatenelement jeder Gruppe `ip.src` .
3. Die normalen, nicht aggregierten Metadatenelemente werden nach dem `group key` zurückgegeben, werden aber alle dieselbe Gruppennummer aufweisen wie die Gruppenschlüsselmetadaten.
4. Danach werden die aggregierten Ergebnismetafelder für jede Gruppe zurückgegeben.
5. Alle Felder innerhalb einer Gruppe werden zusammen zurückgegeben. Verschiedene Gruppenergebnisse werden sich nicht überlappen.

Wenn einem der `group by` -Metadatenelemente von einer der durch die `where` -Klausel zugeordneten Sitzungen fehlen, wird das entsprechende Metafeld für die Zwecke dieser Gruppe als NULL behandelt. Wenn die Ergebnisse für diese Gruppe zurückgegeben werden, werden die als NULL gewerteten Teile des Gruppenschlüssels aus den Gruppenergebnissen ausgelassen, da die Datenbank kein Konzept von NULL hat.

Die Semantik einer `group by` -Abfrage unterscheidet sich von einer SQL-artigen Datenbank dahingehend, welche Metafelder zurückgegeben werden. SQL-Datenbanken erfordern, dass Sie die `group by` -Spalten explizit in der `select` -Klausel auswählen, wenn Sie möchten, dass sie im Ergebnissatz zurückgegeben wird. Die NetWitness Core-Datenbank gibt die Gruppenspalten immer implizit zuerst zurück.

Eine Abfrage mit einer `group by` -Klausel honoriert den Parameter `size` für den Ergebnissatz, falls einer angegeben wurde. Allerdings legt sie aufgrund des Typs der Gruppierung dem Aufrufer eine zusätzliche Belastung auf, Gruppen zu paginieren und zu reformieren, wenn ein Ergebnissatz mit fester Größe gefordert wird. Aus diesem Grund sollten Sie keine ausdrückliche Ergebnisgröße angeben, wenn Sie einen `group by` -Aufruf tätigen. Wenn Sie keine ausdrückliche Größe angeben, wird der gesamte Ergebnissatz als teilweises Ergebnis geliefert.

Die folgende Tabelle beschreibt die Datenbank-Honorierungs-Konfigurationsparameter, die die Auswirkung von I/O oder Speicher einer Group-by-Abfrage beschränken.

Parameter	Funktion
`/sdk/config/max.query.groups`	Dies ist der Grenzwert für die Anzahl der Gruppen, die im Speicher gehalten werden können, um Aggregate zu berechnen. Dieser Parameter erlaubt Ihnen, die Gesamtspeichernutzung der Abfrage zu begrenzen.
`/sdk/config/max.where.clause.sessions`	Dies ist der Grenzwert für die Anzahl der Sitzungen, die von der Where-Klausel in einer Abfrage verarbeitet werden können. Dieser Parameter erlaubt Ihnen, einen Grenzwert für die Anzahl der Sitzungen einzustellen, die von den Metadaten- und Sitzungsdatenbanken gelesen werden müssen, um eine Abfrage zu lösen.

`order by` -Klausel (seit 10.5)

Eine `order by` -Klausel kann zu einer Abfrage hinzugefügt werden, die eine `group by` -Klausel enthält. Die `order by` -Klausel sorgt dafür, dass der Satz gruppierter Ergebnisse in sortierter Reihenfolge zurückgegeben wird.

Eine `order by` -Klausel besteht aus einem Satz von zu sortierenden Elementen in aufsteigender oder absteigender Reihenfolge. Das Sortieren kann auf jedem Datenfeld durchgeführt werden, das im Ergebnissatz zurückgegeben wird. Dazu zählen von der `select` -Klausel spezifizierte Metadaten, von der `select` -Klausel spezifizierte Ergebnisse von Aggregatfunktionen oder `group by` -Metadatenfelder.

Die `order by` -Klausel kann über mehrere Spalten sortieren. Es gibt keinen Grenzwert für die Anzahl zulässiger `order by` -Spalten in der Abfrage, aber es gibt eine praktische Begrenzung insoweit, dass jede der `order by` -Spalten sich auf etwas beziehen muss, das von der `select` -Klausel oder `group by` -Klausel zurückgegeben wurde. Die Sortierung über mehrere Spalten geschieht lexikographisch, das heißt, wenn zwei Gruppen gleiche Werte für die erste Spalte haben, werden sie nach den zweiten Spalten sortiert. Wenn auch die zweiten Spalten gleich sind, werden sie nach den dritten Spalten sortiert usw. über alle `order by` -Spalten hinweg.

Die NetWitness Core-Datenbank ist insofern einzigartig, als dass jede der Ergebnisgruppen, die auf eine Abfrage zurückgegeben werden, viele Werte für eine Auswahl haben können. Zum Beispiel ist es möglich, alle einem Metadatentyp entsprechenden Metadaten auszuwählen und sie in Gruppen zu organisieren und es ist möglich, mithilfe der Funktion `distinct()` Gruppen von unterschiedlichen Metawerten zurückzugeben. Wenn eine `order by` -Klausel eines der Felder in der Gruppe mit mehreren Werten referenziert, wird wie folgt sortiert:

1. Innerhalb jeder Gruppe werden die Felder mit mehreren passenden Werten nach der Sortierklausel sortiert.
2. Alle Gruppen werden sortiert, indem das erste Vorkommnis des sortierten Feldes, das innerhalb jeder Gruppe gefunden wird, verglichen wird.

Die `order by` -Klausel ist nur in Abfragen verfügbar, die eine `group by` -Klausel haben, da Gruppen die Metafelder in unterschiedliche Datensätze organisieren müssen. Wenn Sie eine beliebige Abfrage so sortieren möchten, als wäre keine Gruppierung angewendet worden, verwenden Sie `group by sessionid` . So wird sichergestellt, dass die Ergebnisse in Gruppen von unterschiedlichen Sitzungen oder Ereignissen zurückgegeben werden.

`group by` -Klauseln werden natürlicherweise in aufsteigender Gruppenschlüssel-Reihenfolge zurückgegeben, aber mithilfe einer `order by` -Klausel können Gruppen auch in anderer Reihenfolge zurückgegeben werden.

Wenn bei einer `order by` -Spalte weder `asc` noch `desc` angegeben ist, wird standardmäßig aufsteigend sortiert.

Beispiele:

```
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
```

`values` -Aufruf

Der Index hält eine Funktion für `values` auf niedriger Stufe bereit, um auf die im Index gespeicherten eindeutigen Metawerte zugreifen zu können. Dank dieser Funktion können Entwickler erweiterte Funktionen an Gruppen eindeutiger Metawerte ausführen.

Parametersyntax für den Aufruf `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 ;
```

Mit dem `values` -Aufruf steht eine Funktion für die Rückgabe einer Reihe von eindeutigen Metawerten für einen angegebenen Metaschlüssel bereit. Der `values` -Aufruf kann für jeden eindeutigen Wert eine aggregierte Gesamtanzahl zurückgeben. Die für die Berechnung der Summe verwendete Funktion wird vom Flags-Parameter gesteuert.

Parameter

In der folgenden Tabelle werden die Eigenschaften der einzelnen Parameter beschrieben:

Parameter	Funktion
`fieldName`	Dies ist der Name des Metaschlüssels, für den Sie eindeutige Werte abrufen. Wenn `fieldName` beispielsweise `ip.src` lautet, gibt diese Funktion die eindeutigen Quellen-IP-Werte in der Sammlung zurück. Wenn sich `fieldName` auf einen Schlüssel mit Umbenennungsverweisen bezieht, wird das Ergebnis als die Kombination von Feldwerten für den angegebenen Metaschlüsselnamen und allen Metaschlüsseln der Verweise definiert.
`where`	Dies ist eine `where` -Klausel, die der Filterung derjenigen Sitzungen dient, für die eindeutige Werte zurückgegeben wurden. Wenn beispielsweise `fieldName` den Wert `ip.src` aufweist und die `where` -Klausel `ip.src = 192.168.0.0/16` ist, werden nur Werte im Bereich zwischen `192.168.0.0` und `192.168.255.255` zurückgegeben. Weitere Informationen zur Syntax der `where` -Klausel finden Sie unter *Where-Klauseln*.
`size`	Die Größe der zurückgegebenen Menge von Werten. Mit dieser Funktion wird eine kleine Untermenge der möglichen eindeutigen Werte aus der Datenbank zurückgegeben.
`id1` , `id2`	Mit diesen optionalen Parametern wird der Umfang der Suche nach eindeutigen Werten auf einen bestimmten Abschnitt der Metadatenbank und des Index begrenzt. Die Festlegung der Parameter `id1` und `id2` für einen begrenzten Bereich der Metadatenbank ist unabdingbar, um umfangreiche Sammlungen schnell durchsuchen zu können.
`flags`	Mit Flags wird die Sortierung und Aufsummierung von Werten

	gesteuert. Flags werden im folgenden Abschnitt „Werte-Flags“ beschrieben.
`threshold`	Durch Setzen des Parameters `threshold` kann der `values` -Aufruf die Erfassung der mit jedem Wert verknüpften Summe abkürzen, sobald der Schwellenwert erreicht ist. Durch die Angabe eines Schwellenwertes kann der Aufrufer die Menge der aus der Datenbank abzurufenden Index- und Metaelemente beschränken. Diese Optimierung findet nicht statt, wenn der Parameter `threshold` nicht festgelegt oder auf null gesetzt wurde.
`aggregateFunction`	Optionaler Parameter, der zum Ändern des Standardverhaltens vom Zählen von Sitzungen, Paketen oder Größen bis hin zum Zählen oder Aufsummieren des durch `aggregateFieldName` definierten numerischen Felds verwendet wird. Wenn einer der Parameter definiert wurde, müssen beide angegeben werden. Übergeben Sie entweder `sum` oder `count` , um anzugeben, welches Verhalten ausgeführt werden soll.
`aggregateFieldName`	Das Metadatenfeld, für das `aggregateFunction` ausgeführt wird. Sowohl der Parameter `aggregateFunction` als auch der Parameter `aggregateFieldName` müssen angegeben werden, wenn das Flag `aggregate` gesetzt ist. Die Ausführung eines `values` -Aufrufs mithilfe einer der Aggregatfunktionen kann erheblich langsamer erfolgen als ein `values` -Aufruf, der die Summen von Sitzungen, Paketen oder Größen erfasst. Das liegt daran, dass für jede übereinstimmende Sitzung die `where` -Klausel aus der Metadatenbank abgerufen werden muss. Bei diesem Scanvorgang erfolgt eine I/O-Bindung eines überwiegenden Teils der Abfrage an die Meta-Datenbank-Volumes. Die für einen aggregierten `values` -Aufruf erforderliche Zeit verhält sich linear proportional zur Anzahl der Sitzungen, die der `where` -Klausel entsprechen.
`min` , `max`	Der Mindest- und der Maximalwert, die vom Aufruf zurückgegeben werden sollten. Diese Parameter werden für die Iteration (oder das

Paging) einer besonders hohen Anzahl von Werten verwendet – üblicherweise mehr Werte, als von einem einzelnen Aufruf zurückgegeben werden könnten. Sie werden vor allem zusammen mit den Flags `sort-value`,`sort-ascending` verwendet, sodass der höchste zurückgegebene Wert in einem nachfolgenden Aufruf als Wert für den Parameter `min` verwendet wird. Es handelt sich um exklusive Werte. Wenn `min="rsa"` angegeben wurde und `rsa` ein gültiger Wert war, wird nicht `rsa` zurückgegeben, sondern der nächsthöhere Wert.

`values` -Flags

Mit dem Parameter `flags` wird die Funktion des Werteaufrufs gesteuert. Es gibt drei Gruppen von Flags, die den unterschiedlichen Operationsmodi entsprechen, wie Sie der folgenden Tabelle entnehmen können.

Flag	Beschreibung
`sessions`, `size`, `packets`	Mit dem Aufruf `values` können Sie eins der folgenden Flags angeben, um zu bestimmen, wie die Summe für jeden Wert berechnet werden soll. Beim Flag `sessions` gibt der Aufruf `values` eine Anzahl von Sitzungen zurück, die jeden Wert enthalten. Mit dem Flag `size` summiert der Aufruf `values` die Größe aller Sitzungen, die jeweils eindeutige Werte enthalten, und meldet die Gesamtgröße für jeden eindeutigen Wert. Beim Flag `packets` summiert der Werteaufruf die Anzahl der Pakete in allen Sitzungen, die jeweils eindeutige Werte enthalten, und gibt diese Summe für jeden eindeutigen Wert zurück.
`sort-total`, `sort-value`	Mit diesen Flags wird die Sortierung der Flags gesteuert. Wenn das Flag `sort-total` lautet, wird die Ergebnismenge in der Reihenfolge der erfassten Summen sortiert. Mit dem Flag `sort-value` erfolgt die Sortierung der Ergebnisse in der Reihenfolge ihrer Werte.
`order-ascending`, `order-descending`	Mit diesen Flags wird die Sortierreihenfolge der Ergebnismenge gesteuert. Wenn zum Beispiel nach der Summe in absteigender Reihenfolge sortiert werden soll, dann werden die Werte mit der größten Summe zuerst zurückgegeben.

Beispiele für den Aufruf `values`

Der Aufruf `values` wird von der Ansicht „Navigation“ in NetWitness Suite sehr häufig verwendet. Die Standardansicht erzeugt Aufrufe, die wie folgt aussehen:

```
/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\""
```

In diesem Beispiel fordert die Ansicht „Navigation“ eindeutige Werte für `ip.src` an. Es werden eindeutige Werte von `ip.src` im angegebenen Zeitbereich angefordert. Die Anforderung bezieht sich auf die Anzahl von Sitzungen, die mit jedem `ip.src` übereinstimmen. Ausgegeben werden die ersten 20 `ip.src` -Werte bei einer Sortierung nach der Gesamtanzahl der Sitzungen in absteigender Reihenfolge. Außerdem verfügt die Ansicht „Navigation“ über einen Meta-ID-Bereich, um der Suchmaschine einen Optimierungshinweis zu liefern.

`msearch` -Aufruf

Der Index bietet die Low-Level-Funktion `msearch` für die Durchführung von Textsuchvorgängen für alle Metadatentypen. Für diese Art von Suche müssen Benutzer in ihren Abfragen keine bekannten Metadatentypen definieren. Stattdessen werden alle Teile der Datenbank auf Übereinstimmungen durchsucht. `msearch` wird von der Textsuche der Ereignisansicht verwendet. Details zu den akzeptierten Suchformularen und Beispiele finden Sie im Thema „Filter und Suchergebnisse in der Ereignisansicht“ im *Leitfaden zu Investigation und Malware Analysis* .

`msearch` -Parameter:

```
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" ;
```

Der `msearch` -Algorithmus funktioniert folgendermaßen:

1. Ein Satz von Sitzungen wird im Index ermittelt, indem die Schnittmenge der folgenden drei Sätze gesucht wird:

- (Satz 1) Alle Sitzungen in der Datenbank
- (Satz 2) Sitzungen, die dem Parameter der `where` -Klausel entsprechen
- (Satz 3) Wenn das Flag `si` angegeben ist: Sitzungen, die Werte indiziert haben, die dem Suchzeichenfolgen-Parameter entsprechen.

2. Wenn bei der Suche der Parameter `sm` angegeben ist, werden alle Metadaten aus dem im Schritt 1 ermittelten Sitzungssatz gelesen und auf Übereinstimmung mit dem Suchzeichenfolgen-Parameter geprüft. Die Metadatenelemente werden aus dem Service gelesen, der dem Punkt am nächsten liegt, von dem aus die Suche ausgeführt wurde. Wenn die Suche beispielsweise in einem Broker ausgeführt wurde, können die Metadatenelemente aus dem dem Broker nächstgelegenen Concentrator gelesen werden, wenn aber die Suche in einem Archiver ausgeführt wurde, werden die Metadatenelemente aus dem Archiver selbst gelesen.
3. Wenn bei der Suche der Parameter `sp` angegeben ist, werden alle Rohpaketdaten oder Rohdatenprotokolleinträge aus dem im Schritt 1 ermittelten Sitzungssatz gelesen und auf Übereinstimmung mit dem Suchzeichenfolgen-Parameter geprüft. Die Pakete werden aus dem Service gelesen, der dem Punkt am nächsten liegt, von dem aus die Suche ausgeführt wurde. Wenn die Suche beispielsweise in einem Concentrator ausgeführt wurde, werden die Paketdaten aus dem Decoder gelesen, wenn aber die Suche in einem Archiver ausgeführt wurde, werden die Paketdaten aus dem Archiver selbst gelesen.
4. Übereinstimmungen aus Schritt 2 und 3 werden zurückgegeben, sobald sie gefunden werden, bis zum Erreichen des Parameters `limit` . Der limit-Parameter gibt die maximale Anzahl der Sitzungen an, in denen nach Meta- und Paketdatenbanken gesucht wird. Wenn „limit“ nicht angegeben ist, wird der gesamte Satz an Sitzungen durchsucht, der in Schritt 1 bestimmt wurde.

`msearch` -Flags

Flag	Beschreibung
`sp`	Durchsucht Rohpaketdaten
`sm`	Durchsucht alle Metadaten
`si`	Führt vor dem Durchsuchen von Metadaten Indexabfragen für alle Suchparameter aus
`ci`	Führt eine Suche ohne Beachtung der Groß- und Kleinschreibung durch. Bei den zurückgegebenen Ergebnisse wird die Groß- und Kleinschreibung beachtet.
`regex`	Behandelt den Parameter „search“ als regulären Ausdruck. Nur ein regulärer Ausdruck

kann angegeben werden, er kann aber beliebig komplex sein.

`msearch` -Indexsuchmodus

Im Indexsuchmodus, angegeben durch das Flag `si` , werden Ergebnisse deutlich schneller zurückgegeben als in jedem anderen Modus. Die wichtigste Einschränkung dieses Modus ist, dass nur Übereinstimmungen mit Textbegriffen zurückgegeben werden, die werteindizierten Metawerten entsprechen.

- Der Parameter `si` muss mit dem Flag `sm` kombiniert werden. Mit dem Parameter `si` wird angegeben, dass die Suche nur in indizierten Metadaten erfolgt.
- Der Parameter `si` kann bei Suchläufen mit regulären Ausdrücken verwendet werden, es werden jedoch nur textindizierte Werte abgeglichen. IP-Adressen und Zahlen werden mit den regulären Ausdrücken nicht geprüft.

`msearch` Tipps

- Verwenden Sie immer die `where` -Klausel, um einen Zeitbereich für die Suche anzugeben.
- Um nach IP-Adressbereichen zu suchen, geben Sie diese in die `where` -Klausel ein.
- Verwenden Sie den Parameter `limit` , wenn Sie nicht den Indexsuchmodus verwenden. Ohne diesen wird sonst eine sehr große Menge an Daten in der Meta- und Paketdatenbank gelesen.

Gespeicherte Verfahren

Die Aufrufe `query` und `values` halten weitere Suchfunktionen auf niedriger Ebene bereit. Für umfassendere Anwendungsbeispiele sind serverseitig gespeicherte Verfahren vorhanden.

Verwendung von Anführungsstrichen in der Abfragesyntax

Der Abfrageparser unterscheidet nicht zwischen einfachen oder doppelten Anführungsstrichen in einer Abfrageanweisung. Ein Wert in einfachen oder doppelten Anführungszeichen wird als Textmetawert behandelt.

Der Abfrageparser versucht, die Bedeutung dessen zu verstehen, was Sie in die Anweisung schreiben. Er ist nicht sehr streng in dieser Hinsicht.

Beispiel:

```
reference.id=4752
```

Diese Klausel identifiziert Sitzungen, die den Metawert `reference.id` und einen *numeric*Wert von 4752 aufweisen.

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

Diese Klausel identifiziert Sitzungen, die den Metawert `reference.id` und einen *string*Wert von `4752` aufweisen.

Allerdings vergleicht die Abfrage-Engine implizit Zahlen und Zeichenfolgen, die wie Zahlen aussehen, und behandelt sie gleich, wenn die Werte semantisch gleich sind. Es funktioniert also mit jeder dieser Schreibweisen.

Für eine möglichst effiziente Performance wäre es allerdings eine gute Idee, die Abfragen so zu konstruieren, dass die Abfragesyntax den vom Parser erzeugten Datentypen entspricht.

Wenn also zum Beispiel der Parser `reference.id` als einen numerischen Datentyp erstellt, (wie `uint32` oder `uint64`), verwenden Sie die numerische Syntax.

Wenn der Parser `reference.id` als einen Textdatentyp erstellt, sollten Sie die Zeichenfolgensyntax verwenden.

Indexanpassung

In diesem Thema wird beschrieben, wie Sie mithilfe der angepassten Indexdatei den Index anpassen können. Jeder NetWitness NextGen-Service wird mit einer Standardindexkonfiguration installiert, die die Indexanforderungen der meisten Benutzer des Produkts erfüllen soll. Es ist allerdings möglich, neue Metaschlüssel zu indizieren, um den Index mit angepasstem Content zu verwenden, der angepasste Metadaten erzeugt hat.

Speicherorte der Indexkonfigurationsdatei

Die Indexanpassung erfolgt, indem an der angepassten Indexdatei Änderungen vorgenommen werden. Der Speicherort dieser Datei ist **/etc/netwitness/ng/index-<servicename>-custom.xml** Dabei gilt Folgendes: **<servicename>** entspricht dem Namen des Produkts, das Sie anpassen. Zum Beispiel ist die angepasste Indexdatei des Concentrator /etc/netwitness/ng/index-concentrator-custom.xml.

Concentrator-Produkte enthalten auch eine Datei, die die Standardindexkonfiguration beschreibt: **/etc/netwitness/ng/index-concentrator.xml**. Diese Datei ist als Vorlage nützlich, die zeigt, wie die angepasste Indexdatei formatiert ist.

Wenn Sie Anpassungen an dem Index in der angepassten Indexdatei vornehmen, setzen diese Anpassungen jeden Konflikt mit der Standardindexkonfiguration außer Kraft.

Sie können Änderungen an der angepassten Indexdatei vornehmen, während der Service noch ausgeführt wird. Wenn der Service einen Befehl zum Speichern des Index empfängt, werden die Änderungen an der angepassten Indexdatei gelesen und auf den Index angewendet.

Änderungen am Index können nur auf neu eingehende Daten angewendet werden. Daten können nicht im Nachhinein mit einer neuen angepassten Indexkonfiguration neu indiziert werden, außer durch einen erneuten Aufbau des Index.

Indexkonfigurationseinträge

Die angepasste Indexdatei ist ein XML-Dokument. Das Stammelement dieses Dokuments ist das Element `language`, in dem sich ein Element pro Metaschlüssel befindet, um die einzelnen angepassten Indizes yu beschreiben. Jedes Element der angepassten Indexkonfiguration sieht wie folgt aus:

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

Definitionen für jedes Attribut in diesem Element: Attribut | Beschreibung -|- name | Der Name des Metaschlüssels, der indiziert wird description | Eine für Menschen lesbare Beschreibung für die Metadaten type level | Der Indextyp, der für diesen Metaschlüssel erstellt wird valueMax | Die maximal eindeutigen Werte, die für diesen Schlüssel pro Scheibe gespeichert werden format | Das Format der Daten, das alle Metaelemente mit diesem Metaschlüsselnamen aufweisen.

In den nächsten Abschnitten werden diese Parameter detaillierter untersucht.

Metanamen

Der vom Index verwendete Metaname bezieht sich auf den Namen des Metaschlüssels, der in jedem Metaelement in der Metadatenbank vorhanden ist. Diese Metanamen werden von den Decodern beim Parsen erzeugt. Parser können entscheiden, Metadaten mit jedem beliebigen Metaschlüsselnamen zu erzeugen. Daher erlaubt Ihnen der angepasste Index, auszuwählen, welche der vom Decoder erzeugten Metaelemente indiziert werden.

Metaschlüsselnamen können 16 Zeichen lang sein und enthalten nur Buchstaben oder das Zeichen „.“.

Datentypen

Wenn der Decoder Metaelemente erzeugt, weist er einen Datentyp zu. Jeder Parser kann den Datentyp der Metadaten, die er erzeugt, selbst auswählen. Es gibt allerdings empfohlene und Standarddatentypen für jeden der Standardmetaschlüssel. Zum Beispiel werden ip.src und ip.dst als der IPv4-Metadatentyp gespeichert und alias.host wird als Textmetadatentyp gespeichert. Jeder Parser muss das Datenformat für jeden vom Decoder erzeugten Metaschlüssel festlegen.

Wenn ein angepasster Index zum Concentrator hinzugefügt wird, muss der Datentyp des angepassten Index mit dem Format der vom Decoder erzeugten Daten übereinstimmen. Wenn die Typen nicht übereinstimmen, versucht der Concentrator, die erzeugten Metadaten in den Typ zu konvertieren, der für den angepassten Index angegeben wurde. Allerdings schlagen diese Konvertierungen manchmal fehl und der resultierende Index kann undefinierte Ergebnisse produzieren.

Ebenso gilt: Wenn viele Decoder und Concentrator als Teil einer NetWitness-Installation zusammen eingesetzt werden, müssen sie sich auf die Typen für jeden Metaschlüssel einigen. Konflikte zwischen NetWitness NextGen-Services in Bezug auf Metadatentypen können zu undefiniertem Verhalten führen.

In der folgenden Tabelle sind die Metadatentypen geyeigt, die von den NetWitness NextGen-Services unterstützt werden.

Typ	Größe in Byte	Beschreibung
Int8	1	Signierte 8-Bit-Ganzzahl
UInt8	1	Unsignierte 8-Bit-Ganzzahl
Int16	2	Signierte 16-Bit-Ganzzahl

UInt16	2	Unsignierte 16-Bit-Ganzzahl
Int32	4	Signierte 32-Bit-Ganzzahl
UInt32	4	Unsignierte 32-Bit-Ganzzahl
Int64	8	Signierte 64-Bit-Ganzzahl
UInt64	8	Unsignierte 64-Bit-Ganzzahl
UInt128	16	Unsignierte 128-Bit-Ganzzahl
Float32	4	32-Bit-Gleitkommazahl, einfache Genauigkeit
Float64	8	64-Bit-Gleitkommazahl, doppelte Genauigkeit
Zeit t	8	Unix epoc Zeitstempel
Binär	1–255	Arbiträre Binärdaten
Text	1–255	UTF-8 Codierte Textdaten
IPv4	4	IPv4-Adressbyte
IPv6	16	IPv6-Adressbyte
MAC	6	MAC-Adressbyte

Beim Definieren eines angepassten Index ist es wichtig, den besten Datentyp für die Metadaten zu verwenden. Speichern Sie zum Beispiel niemals IP-Adressen als Text, da die Textrepräsentation mehr Byte benötigt als die IPv4-Repräsentation.

Indexlevel

Es gibt drei Level, oder Typen, der Indexierung: IndexNone, IndexKeys und IndexValues.

IndexNone

Dieser Typ eines angepassten Index ist gar kein richtiger Index. Angepasste Indexeinträge mit dem Level IndexNone bestehen nur, um den Metaschlüssel zu definieren und zu dokumentieren. IndexNone-Einträge können in angepassten Decoder-Indexen verwendet werden, um einen spezifischen Datentyp für einen Metaschlüssel über alle Parser auf einem Decoder durchzusetzen.

IndexKeys

Dieser Typ eines angepassten Index zeigt an, dass der Index nur solche Sitzungen nachverfolgt, die Metaelemente mit diesem Metaschlüsselnamen enthalten. Allerdings indiziert er keine eindeutigen Werte in der Metadatenbank für den Metaschlüssel.

Key-Level-Indexe lassen sich mit viel weniger Speicherplatz, Arbeitsspeicher und CPU-Zeit managen, aber sie erfordern viel mehr Arbeit von der Abfrage-Engine, wenn Sie mit ihnen Abfrage- oder Wertoperationen durchführen.

Wenn er in einer Where-Klausel verwendet wird, kann ein auf dem Key-Level indexierter Metaschlüssel nur verwendet werden, um Operationen wie „existiert“ oder „!existiert“ zu lösen.

IndexValues

Dieser Typ von angepasstem Index hält Sitzungen, die jeden einzelnen eindeutigen Wert für den Metaschlüssel enthalten. Dieser Indextyp ist auch als „vollständiger Index“ bekannt.

Dieser Indextyp ist erforderlich für die effiziente Verarbeitung der meisten Where-Klauseln und für die Verwendung dieses Metaschlüssels als fieldName-Parameter eines Wertaufrufs.

Value Max

Value max ist ein Parameter, der einen sehr signifikanten Einfluss auf die Genauigkeit und Performance eines Wert-Level-Index haben kann.

Während ein Decoder Pakete oder Protokolle parst, ist es erlaubt, Metadaten jeden Typs mit jedem Wert zu erstellen. Normalerweise werden diese Metaelemente aus Daten erstellt, die direkt aus dem Paket oder Protokoll kopiert wurden. Daher kann jeder eindeutige Metawerte in Reaktion auf beinahe jedes Ereignis erstellen.

Die Performance des Index hängt direkt von der Anzahl eindeutiger Werte ab, die er für jeden Metaschlüssel gefunden hat. Während die Anzahl eindeutiger Werte steigt, kann die Rate, mit der neue Metadaten indexiert werden, abnehmen und die Geschwindigkeit, mit der Abfragen abgeschlossen werden, nimmt ab. Da jede beliebige Person die Erstellung eindeutiger Metawerte beeinflussen kann, ist es für jede beliebige Person möglich, die Performance des Index zu beeinflussen.

Der Parameter value max begrenzt die Anzahl eindeutiger Werte, die in den Index eingehen können. So kann kein böswilliger Benutzer das System mit einer großen Anzahl eindeutiger Werte überfluten, um das NetWitness-System funktionsunfähig zu machen.

Es ist wichtig, einen maximalen Wert für jeden Metaschlüssel einzustellen, dessen Wert möglicherweise direkt durch eingehende Pakete oder Protokolle beeinflusst wird.

Der maximale Wert gilt nur für Werte, die seit dem letzten Speichervorgang des Index hinzugefügt wurden.

Der Grenzwert dafür, wie hoch der maximale Wert eingestellt werden kann, variiert von Version zu Version und hängt von der Menge des verfügbaren RAM für den NetWitness NextGen-Service ab. Für die Version 10.3 liegt der empfohlene obere Grenzwert für value max bei 5.000.000 für jeden Metaschlüssel. Wenn es viele angepasste Indexe gibt, dann muss value max möglicherweise niedriger sein.

maxLength

Der Parameter „maxLength“ wird ausschließlich für den Metadatentyp `word` verwendet. Er muss mit der entsprechenden Einstellung für **/decoder/parsers/config/token.max.length/** im Log Decoder-Service übereinstimmen, der Tokenmetadaten erzeugt. Der Index verwendet „maxLength“, um Suchbegriffe richtig zu interpretieren, die in die SDK-Funktion `msearch` eingegeben werden.

Umbenennen von Schlüsseln

Die Indexsprache unterstützt das Konzept der Umbenennung von Schlüsseln. Diese Funktion wird verwendet, um die Abwärtskompatibilität für neue Schlüsselnamen bereitzustellen, um alte Schlüsselnamen einzustellen und zu ersetzen. Eine Umbenennung wird erreicht, indem Sie `rename`-Elemente zu dem Schlüssel hinzufügen. Damit wird angegeben, dass der übergeordnete Schlüssel den Schlüssel im rename-Element umbenennt. Mit der Schlüsseldefinition unten wird beispielsweise ein neuer Schlüssel mit dem Namen `port_src` definiert, der den Schlüssel `tcp.srcport` umgebenennt.

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

Das rename-Element weist die Datenbank darauf hin, dass bei Verwendung des übergeordneten Schlüssels, in diesem Fall `port_src`, sowohl die Metadatenelemente vom Typ „port_src“ als auch Metadatenelemente vom Typ „tcp.srcport“ eingeschlossen werden. Daher können neue Metadatenelemente mithilfe von `port_src` zur Datenbank hinzugefügt und aus dieser abgefragt werden und solche Abfragen geben Informationen zurück, die zuvor auch in „tcp.srcport“ gespeichert wurden.

Das rename-Element akzeptiert ein einziges Attribut, `name`, das auf einen zuvor definierten Schlüssel verweist.

Schlüssel, auf die von rename-Elementen verwiesen wird, müssen denselben Typ wie der übergeordnete Schlüssel aufweisen.

Schlüssel, auf die von rename-Elementen verwiesen wird, müssen denselben Indexlevel wie der übergeordnete Schlüssel aufweisen.

Wenn ein Schlüssel in einer benutzerdefinierten Indexdatei neu definiert wird und der neu definierte Schlüssel rename-Elemente enthält, ersetzen diese rename-Elemente alle zuvor definierten rename-Elemente.

Neuerstellen des Index

Im normalen Betrieb werden Änderungen an der Indexkonfiguration für einen Service nur auf neue Daten angewendet, die zur Sammlung hinzu kommen. Die Neuerstellung des Index für alle Daten in der Sammlung ist ein zeitaufwendiger Prozess, da der gesamte Speicher Metadatenbank von der Festplatte gelesen werden muss.

In Version 11.0 und höher ist es möglich, den Index neu zu erstellen, während der Service online ist. Version 11.0-Services erstellen Indizes im Hintergrund, wenn der Service feststellt, dass ein Teil der Sitzungs- und Metadatenbanken nicht indiziert sind.

Aktivieren des Hintergrund-Reindexer

Der Hintergrund-Reindexer wird aktiviert, wenn der Service gestartet wird. Während des Starts prüft der Indexer auf Lücken zwischen Sitzungen, die indexiert sind, und Sitzungen, die in der Sitzungs- und Metadatenbank vorhanden sind. Wenn Lücken gefunden werden, beginnt der Hintergrund-Reindexer mit der erneuten Indizierung der Sitzungs- und Metadatenbank auf dem Service.

Beispiele für Ereignisse, die den Hintergrund-Reindexer aktivieren können:

1. Ein Stromausfall oder ein Absturz ist aufgetreten, durch den das letzte Slice des Index beschädigt wird. Die beschädigten Daten werden beim Start gelöscht, wodurch eine Lücke im Index entsteht.
2. Das Löschen der Indexdaten wird erzwungen. Dies geschieht entweder durch Zurücksetzen eines Index oder durch Löschen von Dateien aus dem Dateisystem.

Steuern des Hintergrund-Reindexer

Der Vorgang des Hintergrund-Indexer wird durch den Konfigurations-Node `/index/config/reindex.enable` gesteuert. Wenn `reindex.enable` auf „true“ festgelegt wird, wird der Reindexer beim nächsten Start des Services ausgeführt. Wenn `reindex.enable` auf „false“ festgelegt wird, wird der Reindexer nicht beim nächsten Start des Services ausgeführt, wird jedoch weiterhin ausgeführt, bis der Service neu gestartet wird.

Algorithmus der Hintergrund-Neuindizierung

Der Betrieb des Hintergrund-Indexer läuft wie folgt ab:

1. Der Index untersucht die Bereiche der Sitzungen, die im Index vorhanden sind, und vergleicht diese mit den Bereichen der Sitzungen, die über gültige Metadaten verfügen.

Etwaige Diskrepanzen zwischen den beiden werden als Lücken angesehen.

2. Die Lücken im Index werden basierend auf dem aktuellen Wert von `/index/config/save.session.count` in Slices unterteilt.
3. Für jedes fehlende Slice wird ein temporärer Index in einem der in `/index/config/index.dir` angegebenen Verzeichnisse erstellt. Die Slices werden in umgekehrter numerischer Reihenfolge neu indiziert. Daher werden die zuletzt erfassten Sitzungen zuerst indiziert.
4. Sobald das Slice vollständig neu indiziert wurde, wird es an seinen gültigen Speicherort im Onlineindex verschoben. Wenn das erneut indizierte Slice zum Warm-Tier gehört, wird es in den Warm-Tier verschoben.
5. Die neu indizierten Daten werden als Teil der Sammlung angezeigt.

Status des Hintergrund-Indexer

Der Status-Node `/index/stats/updater.state` gibt den aktuellen Status des Hintergrund-Reindexer an. Dieser Node gibt entweder `running` , `not running` oder `failed` an. Wenn der Status `failed` ist, überprüfen Sie das Serviceprotokoll, um weitere Informationen zu erhalten.

Auswirkungen auf die Aggregation

Services, die Aggregationen ausführen, nutzen den Index, um Sitzungen nachzuverfolgen, die bereits aggregiert wurden. Wenn der Index nicht genügend Informationen zum Aggregationsbeginn aufweist, wird die Aggregation offline ausgeführt, bis genügend Slices neu indiziert wurden. Während dieser Zeit wird der Aggregationsstatus für das Upstream-Gerät angeben, dass es auf die Aggregation wartet.

Erzwingen einer Neuindizierung

So erzwingen Sie die Neuerstellung des Index auf einem Service:

1. Stellen Sie sicher, dass `/index/config/reindex.enable` auf „true“ festgelegt.
2. Setzen Sie den Index mithilfe der Meldung `reset` auf dem Service zurück. Beispiel: `/concentrator/reset index=1` wird den Service neu starten und alle Indexdateien löschen.
3. Warten Sie, bis der Service neu startet. Die Hintergrund-Neuindizierung wird gestartet.

4. Die zuletzt gesammelten Daten werden für Abfragen verfügbar sein, sobald das Index-Slice für diese Sitzungen neu indiziert wurde.

Optimierungstechniken

In diesem Thema werden Optimierungstechniken für die NetWitness Core-Datenbank beschrieben. Die NetWitness Core-Datenbank ist standardmäßig so eingerichtet, dass sie viele verschiedene Workloads verarbeiten kann. Wie bei jede Datenbanktechnologie kann die Performance stark abhängig sein sowohl von der Art der aufgenommenen Daten als auch der Art der Suchvorgänge, die Benutzer in der Datenbank durchführen.

Schwellenwerte

sind eine nützliche Optimierung, die sich erheblich darauf auswirken können, wie schnell Ergebnisse zur Ansicht „Navigation“ von NetWitness Suite zurückgegeben werden. Schwellenwerte werden auf den `values`-Aufruf angewendet. Weitere Informationen über den `values`-Aufruf erhalten Sie unter Abfragen.

Mit dem Schwellenwert wird begrenzt, wie viel der Datenbank von der Festplatte abgerufen wird, um eine Anzahl zu erzeugen. Bei den meisten Abfragen ist die Anzahl der Sitzungen, die der `where`-Klausel entsprechen, sehr groß. Wenn Sie beispielsweise alle Protokollereignisse nur einer Stunde auswählen, ergibt dies bei 30.000 Ereignissen pro Sekunde 108.000.000 entsprechende Sitzungen.

RSA führte den Schwellenwert aufgrund der Feststellung ein, dass das Ergebnis in den meisten Fällen, in denen eine Sitzungsanzahl erforderlich ist, nicht bis auf die letzte Sitzung exakt sein muss. Wenn z. B. in einer Übersicht erfasst werden soll, welche 20 IP-Adressen in der letzten Stunde am häufigsten auftraten, ist es nicht so wichtig, ob der Bericht für einen IP-Wert 10.000.000 oder 10.000.001 entsprechenden Sitzungen angibt. Eine Schätzung ist ausreichend. In solchen Szenarien kann als Anzahl ein Schätzwert zurückgegeben werden, wenn die Anzahl den Schwellenwertparameter übersteigt. Wenn der Schwellenwert erreicht wird, wird die verbleibende Anzahl geschätzt, und die Ergebnis werden bei Bedarf auf der Grundlage der geschätzten Anzahlen sortiert.

Komplexe `where`-Klauseln

Wie lange es dauert, bis die NetWitness Core-Datenbank ein Ergebnis erzeugt, hängt von der Komplexität der Abfrage ab. Abfragen, die direkt an den Indexen der Metadaten ausgerichtet sind, können schnell beantwortet werden, aber es ist sehr leicht, Abfragen zu schreiben, die nicht schnell beantwortet werden können. Manchmal werden Abfragen, auf die nicht schnell ein Ergebnis zurückgegeben werden kann, von der Core-Datenbank und dem Index unterschiedlich verarbeitet, um dem Kunden ein zufriedenstellenderes Ergebnis zu bieten.

Es ist nützlich, die relativen *Kosten* jedes Teils der `where`-Klausel zu kennen. Die Ausführung einer Klausel mit hohen Kosten dauert länger. In der folgenden Tabelle sind die Abfragevorgänge entsprechend der relativen Kosten geordnet, von den niedrigsten zu den höchsten.

Vorgang	Cost
exists, !exists	Konstant
= , !=	Logarithmisch in Bezug auf die Anzahl der eindeutigen Werte für den Metaschlüssel, linear in Bezug auf die Anzahl der eindeutigen Element, die einem Bereichsausdruck entsprechen
< , > , <= , >=	Logarithmisch in Bezug auf eindeutige Wertsuche, aber wahrscheinlicher linear, da der Ausdruck einem weiten Wertebereich entspricht
`begins` , `ends` , `contains`	Linear in Bezug auf die Anzahl der eindeutigen Werte für den Metaschlüssel
`regex`	Linear in Bezug auf die Anzahl der eindeutigen Werte für den Metaschlüssel mit hohen pro-Wert-Kosten

`AND` und `OR`

Bedenken Sie bei der Erstellung einer `where`-Klausel, dass sich die Aneinanderreihung vieler Begriffe mit dem Operator `AND` positiv auf die Performance der Abfrage auswirken kann. Wann immer der Satz Sitzungen, die der `where`-Klausel entsprechen, durch mehrere Kriterien gefiltert werden kann, ist der Arbeitsaufwand für die Abfrage reduziert. Entsprechend vergrößert jede `OR`-Klausel den Satz Sitzungen, die für eine Abfrage verarbeitet werden müssen.

Als allgemeine Faustregel gilt: Je mehr `AND`-Klauseln eine Abfrage enthält, um so schneller wird sie ausgeführt, aber je mehr `OR`-Klauseln eine Abfrage enthält, um so langsamer sie ausgeführt.

Anwendungsbeispiel: Abgleichen eines großes Subnetz

Benutzer versuchen beim Erstellen von Abfragen häufig, Subnetze der Klasse A ein- oder auszuschließen. Hierbei handelt es sich um eine gängige Abfrageart, da die Benutzer versuchen, einen großen Teil ihres internen Netzwerks in die Ermittlungen einzuschließen oder davon auszuschließen.

Es ist problematisch für die Abfrage-Engine, die Abfrage ausschließlich mit dem Index `ip.src` oder `ip.dst` zu verarbeiten. Das Problem besteht darin, dass eine `where`-Klausel wie diese:

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

Tatsächlich wie folgt interpretiert werden muss:

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

Daher muss der Index unter Umständen eine `where`-Klausel mit über 16 Millionen Begriffen erstellen.

Diese Problem lässt sich mithilfe des Decoder lösen, mit dem gängige, in Frage kommende Netzwerke mit Anwendungsregeln kennzeichnen kann. Beispielsweise könnten Sie Metadatenelemente mit einer Anwendungsregel erstellen, die folgendermaßen aussieht:

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

Mit dieser Regeln werden Metadatenelemente im Metaschlüsselnetzwerk mit dem internen Wert einer beliebigen IP-Adresse im 10.0.0.0/8-Netzwerk erstellt.

Die `where`-Klausel könnte wie folgt ausgedrückt werden:

```
network = "internal"
```

Angenommen, es gibt einen `value-level`-Index für die Netzwerkmetadaten, dann muss diese Abfrage durch den Index nicht in etwas Komplexeres erweitert werden und die Sitzungen, die dem gewünschten Subnetz entsprechen, werden schnell gefunden.

Anwendungsbeispiel: Abgleichen von Teilzeichenfolgen

Die Verwendung der Operatoren `begins`, `ends`, `contains` und `regex` in einer `where`-Klausel kann die Ausführung stark verlangsamen, wenn es eine große Anzahl von Werten für den Metaschlüssel gibt. Jeder dieser Operatoren wird einzeln gegen jeden eindeutigen Wert ausgewertet. Wenn der Operator `regex` lautet, muss `regex` einzeln gegen jeden eindeutigen Wert ausgeführt werden.

Die wirkungsvollste Strategie zur Umgehung ist die Neuorganisation der Metadatenelemente, sodass der Benutzer kein Abgleichen von Teilzeichenfolgen verwenden muss.

Beispiel: Angenommen, die Benutzer versuchen, den Hostnamen innerhalb einer URL irgendwo in der Sitzung zu finden. Die Benutzer könnten eine `where`-Klausel wie folgt schreiben:

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

In diesem Szenario enthält der Metaschlüssel `url` wahrscheinlich einen eindeutigen Wert für jede Sitzung, die vom Decoder erfasst wurde, und daher gibt es eine große Anzahl von eindeutigen Werten. In diesem Fall ist der contains-Vorgang langsam.

Es empfiehlt sich, den Teil der Metadaten zu identifizieren, der abgeglichen werden soll, und den Abgleich dann in den Inhaltsparser zu verschieben.

Wenn beispielsweise Metadaten für jede URL erzeugt werden, könnte ein Parser die URL auch in ihre Bestandteile zerlegen. Wenn der Decoder beispielsweise URL-Metadaten mit dem Wert `http://www.rsa.com/netwitness` erstellt, könnte er auch `alias.host`-Metadaten mit dem Wert `www.rsa.com` erstellen. Abfragen könnten wie folgt formuliert werden:

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

Da der Teilzeichenfolgenoperator nicht mehr erforderlich ist, wird die Abfrage viel schneller durchgeführt.

Indexspeicherung

Der Core-Index wird in Speicherpunkte, auch als Slices bezeichnet, unterteilt. Wenn der Index gespeichert wird, werden alle Daten im Index auf die Festplatte geschrieben und dieser Teil des Index wird als Schreibgeschützt gekennzeichnet. Speicherungen dienen zwei Funktionen:

- Jeder Speicherpunkt stellt einen Platz dar, an dem der Index im Falle eines Stromausfalls wiederhergestellt werden kann.
- Durch regelmäßiges Speichern können Sie sicherstellen, dass der Teil des Index, der aktiv aktualisiert wird, nicht die Größe des RAM übersteigt.

Speicherpunkte partitionieren den Index in unabhängige, nicht überlappende Segmente. Wenn eine Abfrage mehrere Speicherpunkte überschreiten muss, müssen Teile der Abfrage erneut ausgeführt und die Ergebnisse zusammengeführt werden. Dadurch dauert die Ausführung dieser Abfrage länger.

In NetWitness Suite 10.5 und späteren Installationen wird der Core-Index standardmäßig jedes Mal gespeichert, wenn der Datenbank 600.000.000 Sitzungen hinzugefügt wurden. Dieses Intervall wird durch den Indexkonfigurationsparameter `save.session.count` festgelegt. Weitere Informationen finden Sie unter Indexkonfigurations-Nodes.

Ältere Versionen von NetWitness Suite oder Systeme, die von NetWitness Suite-Versionen vor 10.5 aktualisiert wurden, verwenden eine zeitbasierte Speicherplanung, mit der der Index alle 8 Stunden gespeichert wird. Sie können das aktuelle Speicherintervall über den Scheduler-Editor in der NetWitness Admin-Benutzeroberfläche für den Service anzeigen. Der Standardeintrag sieht wie folgt aus:

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

Durch die Anpassung des Intervalls können Sie steuern, wie häufig Speicherungen erstellt werden.

Auswirkungen der Vergrößerung des Speicherintervalls

Durch eine Vergrößerung des Speicherintervalls werden Speicherpunkte weniger häufig erstellt und folglich sind weniger Speicherpunkt vorhanden. Dies wirkt sich positiv auf die Abfrageperformance aus: Es ist unwahrscheinlicher, dass Abfragen über Slices hinweg durchgeführt werden müssen, und wenn dies doch der Fall ist, müssen weniger Slices überquert werden.

Die Vergrößerung des Speicherintervalls weist jedoch auch Nachteile auf. Erstens ist die Wahrscheinlichkeit höher, dass der Concentrator den für einen der Indexe festgelegten Grenzwert `valueMax` erreicht. Zweitens ist die Recovery-Zeit erhöht, wenn das Gerät zwangsweise heruntergefahren werden muss oder wenn ein Stromausfall auftritt. Und drittens können sich Index-Slices, die zu groß für den Speicher sind, nachteilig auf die Aggregationsrate auswirken.

Auswirkungen der Verkleinerung des Speicherintervalls

Durch eine Verkleinerung des Speicherintervalls kann ein Erreichen des Grenzwerts `valueMax` vermieden werden, während gleichzeitig ein vollständiger Werteindex für Metadaten, der eine große Anzahl eindeutiger Werte enthält, erhalten bleibt. Die Verkleinerung des Speicherintervalls beeinträchtigt die Abfrageperformance, da mehr Slices erstellt werden.

Arbeiten mit `valueMax`

Die Beschränkung durch `valueMax` ist für Benutzer frustrierend, die alle möglichen eindeutigen Metadaten indexieren möchten. Leider ist dies im Allgemeinen nicht möglich. Es gibt Metaschlüssel, die willkürliche, zufällige Daten von überall aus dem Internet enthalten können, und alle eindeutigen Werte können nicht indexiert werden.

Es ist jedoch möglich, einige der Beschränkungen von `valueMax` durch die Verwendung von Indexen auf Schlüsselebene anstelle von WertIndexen zu umgehen. `valueMax` hat keine Auswirkungen auf Indexe auf Schlüsselebene.

Es ist möglich, die Ansicht „Navigation“ mit einem Metaschlüssel, der auf Schlüsselebene indiziert ist, zu verwenden. Von der Datenbank werden in der `where`-Klausel nach Möglichkeit Indexe auf Wertebene verwendet, eindeutige Werte werden jedoch durch Metadatenbankscanning für den `values`-Aufruf aufgelöst. Dieser Ansatz funktioniert ausgezeichnet, wenn die `where`-Klausel einen wirksamen Filter bietet, mit dem der Suchumfang auf eine kleine Anzahl von Sitzungen eingeschränkt werden kann, etwa weniger als 10.000 Sitzungen.

Wenn `valueMax` erreicht wird, können Benutzer einen Datenbankscan an ihren Abfragen durchführen, um sicherzustellen, dass keine relevanten Werte verworfen wurden. Auf diese Funktion können Sie im Investigator 9.8-Client über das Kontextmenü in der Ansicht „Navigation“ zugreifen. Obgleich der Metadatenbankscan viel Zeit beansprucht, kann sich der Benutzer damit beruhigen, dass in den Berichten nichts fehlt.

Parallelisieren von Workloads

Wenn der Benutzer viele Berichte verwendet, muss sichergestellt sein, dass die Optionen zur parallelen Ausführung in Reporting Engine voll ausgeschöpft werden. Außerdem muss sichergestellt sein, dass die Anzahl von `max.concurrent.queries` der Hardware angemessen ist.

Die Ansicht „Navigation“ in NetWitness Suite ermöglicht eine parallele Ausführung der Komponenten der Ansicht „Investigator“, was sich erheblich auf die wahrgenommene Performance des NetWitness Core-Service auswirken kann.

Indexwiederherstellung

In seltenen Fällen kann ein Core-Service von einer Indexwiederherstellung profitieren. Beispiele:

- Die Index-Slices in einem NetWitness Core-Service wurden von einer sehr alten Version des Produkts erstellt und er hat seit mehr als 6 Monaten keine Daten bereitgestellt.
- Der Index wurde falsch konfiguriert und der Benutzer möchte alle Metadaten mit einer neuen Indexkonfiguration neu indexieren.
- Die beim Core-Service eingehende Datenverkehrslast war sehr gering und das Speicherintervall war groß, wodurch mehr Slices als erforderlich erzeugt wurden.

In solchen Fällen kann eine Indexwiederherstellung die Performance verbessern. Dazu müssen Sie die Nachricht `reset` mit dem Parameter `index=1` an den Ordner `/decoder` auf einem Decoder, den Ordner `/concentrator` auf einem Concentrator oder den Ordner `/archiver` auf einem Archiver senden.

Beachten Sie, dass die vollständige Indexwiederherstellung eines komplett ausgelasteten Concentrators mehrere Tage dauert und möglicherweise Wochen bei einem kompletten Archiver.

Skalieren der Aufbewahrung

Es gibt mehrere Möglichkeiten, um die Aufbewahrung in der NetWitness Core-Datenbank zu verbessern. Aufbewahrung bezieht sich auf den Zeitraum, der von den in der Datenbank gespeicherten Daten abgedeckt wird.

Als ersten Schritt zur Analyse der Aufbewahrung müssen Sie bestimmen, welcher Teil der Datenbank der einschränkende Faktor in Bezug auf die Aufbewahrung ist. Die Paket-, Meta- und Sitzungs-Datenbanken bieten die Statistiken `packet.oldest.file.time`, `meta.oldest.file.time` und `session.oldest.file.time` im Ordner `/database/stats`, um das Alter der ältesten Datei in der Datenbank anzuzeigen. Der Index liefert die Statistik `/index/stats/time.begin`, um die älteste im Index gespeicherte Sitzungszeit anzuzeigen.

Erhöhen der Paket- und Metadatenaufbewahrung

Der primäre Mechanismus zur Erhöhung der Aufbewahrung in diesen Datenbanken ist das Hinzufügung von mehr Speicherkapazität. Wenn es nicht möglich ist, dem NetWitness Core-Service Speicherkapazität hinzuzufügen, dann kann es erforderlich sein, die Komprimierungsoptionen auf die Paket- und Metadatenbanken anzuwenden, um die Datenmenge zu reduzieren, die von jeder Datenbank geschrieben wird.

Wenn Sie sich um die Metadatenaufbewahrung sorgen, können Sie nicht benötigten Inhalt von dem Decoder entfernen, der Metadaten erzeugt. Viele Parser erzeugen Metadaten, deren langfristige Speicherung durch den Kunden nicht erforderlich ist.

Erhöhen der Indexaufbewahrung

Der Index wird gewöhnlich länger als die Datenbanken aufbewahrt, bei einem komplexen angepassten Index kann die Indexaufbewahrung jedoch kürzer sein. Im Allgemeinen besteht die einfachste Maßnahme darin, nicht benötigte Indexe auf Wertebene aus der angepassten Konfiguration zu entfernen oder einige der standardmäßigen Indexe auf Wertebene mit Indexen auf Schlüsselebene außer Kraft zu setzen.

Es ist auch möglich, den Index durch Hinzufügen von zusätzlichem Indexspeicher zu skalieren. Der Indexspeicher sollte jedoch nur mithilfe von Solid State Drives erweitert werden.

Horizontal skalieren

Ab Version 10.3 können Concentrators und Archiver mithilfe von Gruppenaggregation in einem Cluster zusammengefasst werden. Die Gruppenaggregation ermöglicht es einem einzigen Decoder Sitzungsfeeds mit Lastenausgleich an mehrere Concentrators oder Archiver zu übertragen. Durch Gruppenaggregation kann die Abfrage- und Aggregations-Workload über einen beliebig großen Hardwarepool verteilt werden.

Weitere Informationen finden Sie im Thema „Gruppenaggregation“ im *Bereitstellungsleitfaden*.

Gruppieren von Workloads

Die NetWitness Core-Datenbank funktioniert viel besser, wenn alle Systembenutzer in derselben Datenbankregion arbeiten. Da die Datenbank Daten vom Decoder mithilfe der First-in-First-out-Methode erhält, werden die Daten in der Datenbank in der Regel anhand des Zeitpunkts ihrer Erfassung und Speicherung in Clustern zusammengefasst. Daher funktioniert die Datenbank besser, wenn alle Benutzer an Daten desselben Zeitraums arbeiten.

Es ist jedoch nicht immer möglich, dass alle Benutzer gleichzeitig denselben Zeitraum bearbeiten. Die NetWitness Core-Datenbank kann dieses Anwendungsbeispiel zwar auch bewältigen, wird jedoch langsamer arbeiten, da sie zwischen verschiedenen Zeiträumen im RAM wechseln muss. Es ist nicht möglich, alle Zeiträume gleichzeitig im RAM zu haben. In der Regel passen weniger als 1 Prozent der Datenbank und weniger als 10 Prozent des Index in den RAM.

Damit NetWitness Suite für den Kunden funktioniert, ist es wichtig, dass der Kunde die Benutzer in Gruppen zusammenfasst, die gewöhnlich mit denselben Zeiträumen arbeiten. Beispielsweise könnte es sich bei Benutzern, die eine tägliche Überwachung der aktuellsten Daten durchführen, um eine Gruppe handeln. Eine andere Benutzergruppe führt Abfragen im Rahmen von Ermittlungen durch, die weiter zurück in die Vergangenheit reichen. Und eine dritte Benutzergruppe erstellt Berichte über lange Zeiträume. Wenn versucht wird, alle Gruppen über eine einzige Datenbank anzudienen, kann dies zu Frustration und langen Wartezeiten auf die zu erstellenden Ergebnisse führen. Wenn die verschiedenen Anwendungsbeispiele jedoch auf unterschiedliche Concentrator-Hardware verteilt werden kann, kann die wahrgenommene Performance viel besser sein. In diesem Fall kann es vorteilhaft sein, mehrere Concentrator-Services mit weniger RAM und CPU-Leistung zu verwenden, anstelle eines einzigen großen und teuren Concentrators, der allen Ansprüchen gerecht werden soll.

Cache-Fenster

Berücksichtigten Sie die Reihenfolge der Ereignisse:

1. Um 9:00 Uhr meldet sich der Benutzer „kevin“ an einem Concentrator an und fordert einen Bericht über die letzte Stunde des Erfassungszeitraums an.
2. Der Concentrator ruft Berichte für den Zeitraum zwischen 08:00 und 09:00 Uhr ab.
3. Um 09:02 Uhr meldet sich der Benutzer „scott“ beim selben Concentrator an und fordert ebenfalls einen Bericht über die letzte Stunde des Erfassungszeitraums an.
4. Der Concentrator ruft Berichte für den Zeitraum zwischen 08:02 und 09:02 Uhr ab.

Obwohl beide Benutzer mit dicht beieinander liegenden Zeiträumen arbeiteten, konnte die vom Concentrator durchgeführte Arbeit, um die Berichte für Kevin zu erstellen, nicht an Scott gesendet werden, da die Zeiträume nicht identisch waren. Der Concentrator musste den Großteil der Berichte für Scott neu berechnen.

Die Einstellung `cache.window.minutes` auf dem Node `/sdk` ermöglicht Ihnen eine Optimierung dieser Situation. Wenn sich ein Benutzer anmeldet, bewegt sich der Zeitpunkt, der die aktuellste Datenerfassung repräsentiert, nur in Schritten vorwärts, die der in dieser Einstellung festgelegten Anzahl von Minuten entsprechen.

Nehmen wir beispielsweise an, `/sdk/config/cache.window.minutes` ist 10\. Die erneute Bewertung der oben genannten Aktion ändert die Reihenfolge der Ereignisse.

1. Um 9:00 Uhr meldet sich der Benutzer „kevin“ an einem Concentrator an und fordert einen Bericht über die letzte Stunde des Erfassungszeitraums an.
2. Der Concentrator ruft Berichte für den Zeitraum zwischen 08:00 und 09:00 Uhr ab.
3. Um 09:02 Uhr meldet sich der Benutzer „scott“ beim selben Concentrator an und fordert ebenfalls einen Bericht über die letzte Stunde des Erfassungszeitraums an.
4. Der Concentrator ruft Berichte für den Zeitraum zwischen 08:00 und 09:00 Uhr ab.
5. Um 09:10 Uhr lädt der Benutzer „scott“ erneut die Berichte für die letzte Stunde des Erfassungszeitraums.
6. Der Concentrator ruft Berichte für den Zeitraum zwischen 08:10 und 09:10 Uhr ab.

Da der in Schritt 3 zurückgegeben Bericht in das Cache-Fenster fällt, wird er sofort zurückgegeben. Dadurch erhält Scott den Eindruck, dass der Concentrator sehr schnell arbeitet.

Das bedeutet, dass größere `cache.window` -Einstellungen die wahrgenommene Performance verbessern, auf Kosten kleiner Verzögerungen bis die aktuellsten Daten für die Suche zur Verfügung stehen.

Zeitliche Beschränkungen

Wenn eine Abfrage lange Zeit in einer NetWitness Core-Datenbank ausgeführt wird, dediziert der Core-Service immer mehr CPU-Zeit und RAM, um diese Abfrage schneller abzuschließen. Dies kann sich nachteilig auf andere Abfragen und die Aggregation auswirken. Um zu verhindern, dass Benutzer mit niedrigeren Berechtigungen mehr als ihren Anteil an den Core-Serviceressourcen nutzen, empfiehlt es sich, Abfragen von normalen Benutzern mit zeitlichen Beschränkungen zu versehen.

Anhang A: Statistik

In diesem Thema werden Statistiken zur Überwachung des Systembetriebs beschrieben. Die Core-Services bieten zahlreiche Statistiken zur Überwachung des Systembetriebs. Einige sind nützlich zur Überwachung der Performance, andere zur Überwachung der Operationen im System oder zum Debugging.

Statistiken in /database/stats

In der folgenden Tabelle wird die Bedeutung der Statistiken in **/database/stats** erläutert.

Statistik	Bedeutung
`meta.bytes`, `packet.bytes`, `session.bytes`	Die Gesamtmenge der Daten (in Byte), die in der jeweiligen Datenbank gespeichert sind
`meta.first.id`, `packet.first.id`, `session.first.id`	Die erste Meta-ID, Paket-ID bzw. Sitzungs-ID, die in der Datenbank gespeichert ist
`meta.last.id`, `packet.last.id`, `session.last.id`	Die letzte Meta-ID, Paket-ID bzw. Sitzungs-ID, die in der Datenbank gespeichert ist
`meta.oldest.file.time`, `packet.oldest.file.time`, `session.oldest.file.time`	Das Erstellungsdatum der ältesten Datei in der jeweiligen Datenbank
`meta.rate`, `packet.rate`, `session.rate`	Die Anzahl der Meta-, Paket- und Sitzungsobjekte, die der jeweiligen Datenbank in der letzten Sekunde hinzugefügt wurden
`meta.total`, `packet.total`, `session.total`	Die Gesamtzahl der Meta-, Paket- und Sitzungsobjekte innerhalb der jeweiligen Datenbank
`meta.volume.bytes`, `packet.volume.bytes`, `session.volume.bytes`	Die ungefähre Volume-Gesamtgröße (in Byte) für alle Verzeichnisse, die von der jeweiligen Datenbank verwendet werden
`meta.free.space`, `packet.free.space`,	Die ungefähre Gesamtgröße des ungenutzten Speicherplatzes (in Byte) für alle Verzeichnisse, die

`session.free.space`	von der jeweiligen Datenbank verwendet werden

Statistiken in /index/stats

In der folgenden Tabelle wird die Bedeutung der Statistiken in **/index/stats** erläutert.

Statistik	Bedeutung
`checkpoint.page`, `checkpoint.summary`	Die letzten Objekte, die bei der Erstellung der letzten Indexspeicherung gespeichert wurden (Debugging)
`index.bytes`	Ein ungefährer Wert dafür, wie viel Festplattenspeicherplatz von Indexdateien benötigt wird
`index.last.load.time`	Der Zeitstempel, wann die gegenwärtige Indexkonfiguration von den Indexkonfigurationsdateien geladen wurde
`memory.used`	Ein ungefährer Wert dafür, wie viel Arbeitsspeicher vom Index belegt wird
`page.first.id`, `summary.first.id`	Die erste Seite und das erste Zusammenfassungsobjekt, die im Index gespeichert sind (Debugging)
`page.last.id`, `summary.last.id`	Die letzte Seite und das letzte Zusammenfassungsobjekt, die im Index gespeichert sind (Debugging)
`page.total`, `summary.total`	Anzahl der Seiten und Zusammenfassungen im Index (Debugging)
`session.first.id`	Die ID der ersten Sitzung im Index
`session.last.id`	Die ID der letzten Sitzung im Index
`sessions.since.save`	Die Anzahl der Sitzungen, die derzeit vom aktuellen Indexslice gehalten werden
`values.added`	Die Anzahl der eindeutigen Werte, die dem aktuellen Indexslice hinzugefügt wurden
`slices.total`	Die Anzahl der Slices im Index
`time.begin`	Die ältesten Zeitmetadaten im Index
`time.end`	Die neuesten Zeitmetadaten im Index
`updater.state`	Der Status des Hintergrund-Reindexer

Statistiken in /sdk/stats

In der folgenden Tabelle wird die Bedeutung der Statistiken in **/sdk/stats** erläutert.

Statistik	Bedeutung
`cache.window.time.begin`	Der Beginn der aktuellen Zeit, die von cache.window.minutes erzwungen wird
`cache.window.time.end`	Das Ende der aktuellen Zeit, die von cache.window.minutes erzwungen wird
`queries.active`	Die Anzahl der Abfragen, die derzeit im Index ausgeführt werden
`queries.queued`	Die Anzahl der Abfragen, die auf die Ausführung warten
`values.calls`	Die Anzahl der Aufrufe der Funktion „values“ seit dem Start des Prozesses
`values.calls.cached`	Die Anzahl der Aufrufe der Funktion „values“, die durch den Werteaufruf-Ergebniscache gelöst wurden

Statistiken pro Abfrage

SDK-Vorgänge wie Abfragen und Werte stellen in `/sdk/config/stats/queries/<handleid>` Informationen über ihren Ausführungsstatus zur Verfügung. Dabei gilt Folgendes: `<handleid>` ist eine eindeutige ID für den Abfragevorgang.

In der folgenden Tabelle sind die Bedeutungen der Statistiken pro Abfrage erläutert.

Statistik	Bedeutung
channel.path	Diese Statistik stellt einen Link zu dem Verbindungskanal bereit, über den der Vorgang kommuniziert. Dieser Kanal dient dazu, Ergebnisse zurück an den Client zu senden.
`query.type`	Die Art des durchgeführten Vorgangs, z. B. Abfragen oder Werte.
`query`	Der vollständige Satz der Parameter, die an die Abfrage übergeben werden.
`query.progress`	Der Prozentsatz der Abfrageausführung, der bereits abgeschlossen

	wurde.
`query.status`	Eine Meldung, die beschreibt, welcher Teil der Abfrageausführung derzeit läuft.
`running.since`	Die Zeit, zu der die Ausführung der Abfrage begonnen hat.
`user`	Der Name des Benutzers, der die Abfrage ausgeführt hat.

Anhang B: Index-Inspektion

Der NetWitness Core-Datenbankindex verfügt über eine integrierte Debuggingfunktion namens `inspect`, die detaillierte Informationen über die Zusammensetzung ihrer Indexe bereitstellt. Die Funktion zur Indexuntersuchung befindet sich unter `/index/inspect` in jeder Core-Service-Konfigurationsstruktur. Services, die keinen Index haben, wie etwa Broker, haben die Funktion `/index/inspect` nicht.

Parameter

Optionen

Typ: Dieser Parameter kann auf den Wert `all-slices` festgelegt werden, um Inspektionsinformationen über jeden Slice im Index zu sammeln. Wenn er nicht festgelegt ist, werden Informationen über den aktuellen, zuletzt erstellten Slice zurückgegeben.

Das Sammeln von Informationen über alle Slices kann sehr lange dauern, wenn der Index viele Slices hat.

Antwort

Die Inspektion gibt viele Zeilen von Schlüsselwertpaaren zurück, die den Status des Index repräsentieren.

Slice-Zusammenfassung

Die erste zurückgegebene Zeile für jeden Slice ist eine Zusammenfassung mit den folgenden Werten.

`session1`	Die erste im Slice indizierte Sitzungs-ID
`session2`	Die letzte im Slice indizierte Sitzungs-ID
`meta1`	Die erste Meta-ID in der ersten im Slice indizierten Sitzung
`meta2`	Die letzte Meta-ID in der letzten im Slice indizierten Sitzung

Zusammenfassung pro Index

Es werden für jeden Index Zeilen mit einer Zusammenfassung pro Index zurückgegeben. Nur Indexe auf Wertebene werden berichtet.

`key`	Der Metaschlüsselname für den Index
`pathname`	Der Pfad auf dem Laufwerk zu diesem Index

`values` Die Anzahl eindeutiger Werte, die in diesem Index gespeichert sind

`summaries` Die Anzahl der zusammenfassenden Einträge, die von diesem Index in der Datei summary.db belegt sind

`pages` Die Anzahl der Seiteneinträge, die von diesem Index in der Datei page.db belegt sind

`sessions` Die Anzahl der Sitzungen, die einen Wert hatten, der in diesen Index eingetragen wurde

`size` Die kumulative Größe aller Metawerte für alle Sitzungen, die einen Wert in diesen Index eingetragen haben

`packets` Die kumulative Größe aller Pakete für alle Sitzungen, die einen Wert in diesen Index eingetragen haben

`summary1` Die erste von diesem Index verwendete Zusammenfassungs-ID

`summary2` Die letzte von diesem Index verwendete Zusammenfassungs-ID

`session1` Die erste von diesem Index referenzierte Sitzungs-ID

`session2` Die letzte von diesem Index referenzierte Sitzungs-ID

Slice-Zusammenfassung Fußzeile

Die letzte Zeile in jedem Inspektionsbericht enthält kumulative Statistiken für alle Indexe im Slice.

`totalKeys` Die Anzahl indizierter Metadatentypen

`totalValues` Die Anzahl eindeutiger Werte, die von allen Indexen in diesem Slice nachverfolgt werden

`totalMemory` Eine ungefährer Gesamtwert für den Speicher, der erforderlich ist, um diesen Index-Slice zu öffnen