RSA NETWITNESS® SUITE

Guide d'optimisation de base de données principale

pour la version 11.0

Informations de contact

RSA Link à l'adresse https://community.rsa.com contient une base de connaissances qui répond aux questions courantes et fournit des solutions aux problèmes connus, de la documentation produit, des discussions communautaires et la gestion de dossiers.

Marques commerciales

Pour obtenir la liste des marques commerciales de RSA, rendez-vous à l'adresse suivante : france.emc.com/legal/emc-corporation-trademarks.htm#rsa.

Contrat de licence

Ce logiciel et la documentation qui l'accompagne sont la propriété d'EMC et considérés comme confidentiels. Délivrés sous licence, ils ne peuvent être utilisés et copiés que conformément aux modalités de ladite licence et moyennant l'inclusion de la note de copyright ci-dessous. Ce logiciel et sa documentation, y compris toute copie éventuelle, ne peuvent pas être remis ou mis de quelque façon que ce soit à la disposition d'un tiers.

Aucun droit ou titre de propriété sur le logiciel ou sa documentation ni aucun droit de propriété intellectuelle ne vous est cédé par la présente. Toute utilisation ou reproduction non autorisée de ce logiciel et de sa documentation peut faire l'objet de poursuites civiles et/ou pénales.

Ce logiciel est modifiable sans préavis et ne doit nullement être interprété comme un engagement de la part d'EMC.

Licences tierces

Ce produit peut inclure des logiciels développés par d'autres entreprises que RSA. Le texte des contrats de licence applicables aux logiciels tiers présents dans ce produit peut être consulté sur la page de la documentation produit du site RSA Link. En faisant usage de ce produit, l'utilisateur convient qu'il est pleinement lié par les conditions des contrats de licence.

Remarque sur les technologies de chiffrement

Ce produit peut intégrer une technologie de chiffrement. Étant donné que de nombreux pays interdisent ou limitent l'utilisation, l'importation ou l'exportation des technologies de chiffrement, il convient de respecter les réglementations en vigueur lors de l'utilisation, de l'importation ou de l'exportation de ce produit.

Distribution

EMC estime que les informations figurant dans ce document sont exactes à la date de publication. Ces informations sont modifiables sans préavis.

février 2018

Sommaire

Présentation de la base de données NetWitness Core

Cette rubrique présente la base de données NetWitness Core. Les services NetWitness Core contiennent une base de données propriétaire développée spécifiquement à des fins d'utilisation dans les produits NetWitness Suite. Elle ressemble peu aux bases de données relationnelles traditionnelles, et elle ne se base sur aucune technologie de base de données du commerce. Ainsi, de nombreux utilisateurs trouvent que la courbe d'apprentissage est raide en vue de comprendre comment la base de données Core fonctionne et comment optimiser son utilisation. Ce guide vise à aider les utilisateurs de NetWitness Suite à comprendre la base de données et à en optimiser le potentiel d'utilisation.

En tant qu'administrateur système, vous pouvez utiliser ces informations pour contribuer à la planification de votre déploiement NetWitness Suite, et pour effectuer un réglage en vue d'optimiser la performance. En tant qu'analyste, vous pouvez utiliser ce guide pour structurer votre analyse de manières qui offriront un retour plus rapide des rapports. En tant que développeur de contenu, vous pouvez utiliser ce guide pour faciliter l'écriture d'un contenu qui sera traité efficacement par le système de base de données.

Produits NetWitness Suite couverts par ce guide

Ce guide couvre les fonctions de NetWitness Suite 11.0. Les composants suivants de NetWitness Suite contiennent la base de données principale :

- Concentrator
- Archiver
- Decoder
- Log Decoder
- Workbench

Termes fréquemment utilisés

Une définition de termes qui sont utilisés dans ce document est présentée ici. Ces termes sont répertoriés dans l'ordre dans lequel ils entrent dans le système NetWitness Suite :

- **Base de données de paquets** : La base de données de paquets contient les données brutes capturées. Sur un Decoder, la base de données de paquets contient les paquets tels qu'ils sont capturés sur le réseau. Les Log Decoders utilisent la base de données de paquets pour stocker

les logs bruts. Les données brutes stockées dans la base de données de paquets sont accessibles par ID de paquet. Toutefois, cet ID n'est généralement pas visible pour l'utilisateur final.

- **ID de paquet** : Nombre utilisé de manière unique pour identifier un paquet ou se connecter à une base de données de paquets.
- **Métabase de données** : La métabase de données contient des éléments d'information extraits par Decoder ou Log Decoder à partir du flux de données brutes. Les analyseurs, les règles et les sources peuvent générer des métaéléments.
- **ID Méta** : Nombre utilisé de manière unique pour identifier un métaélément dans la base de données méta.
- **Clé méta** : Nom utilisé pour classer le type de chaque métaélément. Les méta clés communes incluent les suivantes : ip.src, time ou service.
- **Métavaleur** : Chaque métaélément contient une valeur. La valeur est ce que génère chaque analyseur, source ou règle.
- **Base de données de session** : La base de données de sessions contient des informations qui lient le paquet et les métaéléments en sessions.
- **Session** : Sur un Packet Decoder, une session correspond à un seul flux réseau logique. Par exemple, une connexion TCP/IP représente une session. Sur un Log Decoder, chaque événement de log représente une session. Chaque session contient les références à l'intégralité des ID de paquets et de méta concernés.
- **ID de session** : Nombre unique utilisé pour identifier une entrée de la base de données de sessions.
- **Index** : L'index est une collection de fichiers permettant de rechercher des ID de session à l'aide de métavaleurs.
- **Base de données Core** : Cette base rassemble des paquets, des métas, des sessions et des index.

Pour les définitions de syntaxe, ce document utilise les définitions de grammaire EBNF .

Historique de base de données NetWitness Core

NetWitness (désormais RSA) a développé la base de données principale à des fins d'utilisation dans les systèmes de capture de paquets. Tôt dans l'histoire de NetWitness, les développeurs ont identifié que des technologies de base de données existantes ne seraient pas capables de suivre le taux d'acquisition élevé inhérent à une capture paquet complète. Les technologies contemporaines de base de données étaient loin de pouvoir suivre la capture du nombre de sessions reçues à chaque seconde, et plus encore d'être en mesure de trier chaque paquet. De même, le volume de données signifiait que le stockage des paquets devrait être ignoré et réutilisé aussi vite qu'il était consommé. C'était également une faiblesse des bases de données à l'époque. Ainsi, NetWitness a créé une base de données composées des bases de données de méta, de sessions et de paquets.

Afin de fournir les fonctionnalités analytiques de NetWitness Investigator, un index méta a été ajouté à la base de données NetWitness. L'index a partagé les mêmes objectifs de conception que les bases de données d'origine. Il a été conçu pour supporter une vitesse d'insertion très élevée dans un nombre important d'index très grands.

L'index a évolué considérablement au fil des années. Les versions précoces de l'index étaient uniquement capables de fournir des estimations résumées sur la présence du nombre de métavaleurs uniques dans la base de données méta. D'autres versions ont relevé de grands défis en atteignant des performances de requête acceptables. Par exemple, NetWitness 9.0 mesurait plus fréquemment des temps de rapport en minutes plus qu'en secondes. La version actuelle de l'index provient de l'index NetWitness 9.0, mais elle a considérablement évolué afin de répondre aux attentes en matière de performance et d'ajouter de nouvelles fonctionnalités.

Atouts et faiblesses de la base de données principale

Atouts :

- Vitesse d'insertion soutenue élevée, sans avoir besoin de délai d'inactivité pour les insertions en bloc.
- Performance de requête correcte associée à une vitesse d'insertion élevée.
- Nettoyage ou transfert automatique des anciennes données avec fragmentation minimale.
- Nombre extrêmement élevé d'index de métavaleurs : plus de 100 activés par défaut sur un Concentrator.
- Capacité à dimensionner vers des tailles de base de données en pétaoctets et des tailles de l'index en téraoctets dans un seul nœud.

- En utilisant des paires clé-valeur de méta, le stockage d'éléments métas arbitraires est très flexible dans une session. Ainsi, une session peut être utilisée pour représenter presque toute sorte d'enregistrement de données.

Faiblesses :

- La fonctionnalité de la requête est limitée et de faible niveau.
- Le schéma de base de données de paquet, méta et session est fixe et toute la personnalisation est réalisée via des valeurs et des clés de métas.
- La base de données ne fournit aucune garantie d'atomicité de transaction comme vous pouvez en attendre dans une base de données SQL.

Configuration basique de la base de données

Cette rubrique traite des paramètres de configuration de base de la base de données des services NetWitness Core. Pour plus d'informations sur la configuration des services Core en modifiant les fichiers de configuration, consultez la section « Paramètres de configuration des services » dans le *Guide de mise en route de l'hôte et des services* .

Ce document suppose que le lecteur connaît bien le réglage de la configuration d'un service NetWitness Core. Pour utiliser ce document, vous devez être familier avec l'un des mécanismes de modification de l'arborescence des services Core. Les exemples de ces mécanismes comprennent la vue Explorateur des pages d'administration au sein de l'interface utilisateur NetWitness Suite, ou l'interface REST accessible sur chaque service via un navigateur web.

Obtenir de l'aide au sein du Service Core

Chaque élément de configuration dans un service Core est doté d'une description d'aide intégrée sur la fonctionnalité de l'élément. Vous pouvez afficher ces informations d'aide en plaçant votre souris sur l'élément de configuration dans la vue Explorateur. Chaque élément de configuration indique également s'il peut être changé sans redémarrer ou si un redémarrage du service est nécessaire pour que la modification prenne effet.

Les développeurs utilisant l'API REST peuvent récupérer le texte d'aide pour chaque élément de configuration en envoyant le message `help` au chemin du nœud de configuration.

Stockage des paquets, métas et sessions

Chacune des bases de données de paquets, métas et sessions sont configurées via le dossier `/database/config` sur chaque service NetWitness Core. Chaque base de données dispose d'un paramètre configurable pour spécifier l'emplacement où le service Core stocke les données. Les bases de données des paquets, métas et sessions suivent un schéma prévisible pour l'ensemble de leurs entrées de configuration. Les éléments de configuration de la base de données des paquets commencent par le préfixe `packet` , la configuration de la base de données des métas commence avec le préfixe `meta` et les éléments de configuration de la base de données des sessions commencent par le préfixe `session` .

Stockage d'index

La configuration d'index est stockée dans le dossier `/index/config` sur chaque service Core.

Rubriques

Stockage de base de données hiérarchisé

Cette section décrit le stockage de base de données hiérarchisé et émet des recommandations pour le stockage hiérarchisé Chaud, Actif et Inactif.

À partir de la version 10.4, le service Archiver peut être configuré pour utiliser le stockage hiérarchisé. Le concept du stockage hiérarchisé consiste à placer les données les plus récentes sur un niveau Chaud, qui est le stockage le plus rapide disponible sur le service Archiver.

Tous les services utilisent le niveau Chaud par défaut.

Le niveau suivant dit Actif est généralement moins cher et plus lent, tel qu'un stockage rattaché au réseau. Le niveau Actif contient des données plus anciennes. L'âge dépend de la quantité de stockage qui est allouée au niveau Chaud et du taux d'acquisition moyen. Lorsque le niveau Chaud atteint l'utilisation maximale, la progression normale est de déplacer les données les plus anciennes du niveau Chaud vers le niveau Actif. Dans le cas d'une configuration correcte, cela se produit automatiquement et est transparent pour l'utilisateur final. L'accès aux requêtes et aux données s'effectue automatiquement, quel que soit le niveau (Chaud ou Actif). Cependant, il peut y avoir un impact sur les performances lors de l'accès aux données au niveau Actif par rapport au niveau Chaud, car les temps d'accès au niveau Actif sont généralement plus lents.

En plus des niveaux Chaud et Actif, il existe également un niveau Inactif. Le niveau Inactif est uniquement utilisé comme zone de transit pour la sauvegarde hors ligne. Les services NetWitness Core n'accèdent pas aux données du niveau Inactif. Les services NetWitness Core déplacent les données les plus anciennes vers le niveau Inactif et les considèrent comme étant abandonnées (le service n'utilisera plus les données). Ces données peuvent ensuite être sauvegardées sur un stockage à long terme, comme une bande, durant des mois, voire des années à des fins de restauration éventuelle, en fonction des besoins. La sauvegarde et la suppression ultérieure des données au niveau Inactif doivent être gérées en dehors des services NetWitness Core via des scripts ou d'autres processus.

Si le niveau Inactif est plein parce que les processus externes ne suppriment pas les données à temps, le service NetWitness Core peut ne plus réceptionner les nouvelles données jusqu'à ce que le problème soit corrigé.

Lors du déplacement des données du niveau Inactif, RSA recommande que le répertoire reste sur le même point de montage dans le nouvel emplacement. Par conséquent, si les fichiers proviennent du niveau Actif, il est préférable, pour des raisons de performance, de définir le répertoire de niveau Inactif sur le même système de fichiers. Cela s'explique par le fait que le service tente de déplacer simplement le fichier et le répertoire vers le niveau Inactif, ce qui est une opération quasiment instantanée sur le même système de fichiers. Si le déplacement échoue, l'action de secours consiste à copier les données au niveau Inactif, ce qui engendre plus de temps de traitement et provoque un conflit d'accès E/S supplémentaire sur le niveau à partir duquel il est copié.

Archiver

Les niveaux de stockage sont également utilisés par le service Archiver. Vous pouvez configurer le service Archiver pour utiliser uniquement le stockage Chaud (par défaut), Chaud et Actif ou tous les trois (Chaud, Actif et Inactif). Tous les services doivent utiliser le niveau Chaud. Vous ne pouvez pas configurer un service pour n'utiliser que le niveau Actif. Les données passent du niveau Chaud au niveau Actif pour finir au niveau Inactif. Vous pouvez également ignorer le niveau Actif pour passer directement du niveau Chaud à Inactif. Si le stockage Inactif (hors ligne) n'est pas configuré, les données les plus anciennes sont supprimées au dernier niveau configuré, ce qui est la procédure de fonctionnement standard.

Le déploiement classique du service Archiver définit une taille illimitée pour toutes les bases de données (packet.dir, meta.dir, session.dir, index.dir et éventuellement les variantes de niveau Actif), ce qui signifie que l'indicateur de taille n'est pas pris en compte ou mis à zéro. Cela permet aux bases de données et à l'index de croître sans limite. Au lieu que chaque base de données gère sa propre taille et se déploie uniquement lorsqu'elle dépasse la taille configurée, Archiver les déploie toutes en même temps à l'aide de la commande `/index sizeRoll` . Cela permet aux bases de données et à l'index de se déployer de manière cohérente. Pour plus d'informations sur la commande sizeRoll, reportez-vous à « Transfert asynchrone » dans Transfert .

Archiver est généralement configuré pour placer les bases de données d'index, les bases de données de sessions, les bases de métadonnées et les bases de données de paquets (log) sur le même volume, et non sur plusieurs volumes, comme pour le Concentrator ou le Decoder. Même si cela risque de causer plus de conflits d'accès E/S lorsqu'une lecture simultanée se produit sur plusieurs bases de données, le transfert permet malgré tout d'optimiser la rétention globale. Parce que toutes les bases de données sont sur le même volume, elles sont configurées pour se déployer conjointement, ce qui minimise les données orphelines. Les services Decoder et Concentrator sont configurés pour une vitesse maximale d'E/S, mais peuvent faire l'objet d'une mauvaise estimation du dimensionnement du volume.

Par exemple, si la base de données des sessions est trop grande, il peut y avoir suffisamment de stockage pour six mois de rétention, alors que la base des métadonnées et l'index n'auront que quatre mois de rétention. Parce que la base de données des sessions, la base des métadonnées et l'index sont étroitement liés, la période de rétention la plus courte pour les trois définit la période de conservation globale (dans ce cas, quatre mois). La conservation des bases de données individuelles est principalement affectée par des facteurs hors de contrôle, tels que le trafic capturé, les méta générés (analyseurs, feeds, règles) et le filtrage. Les bases de données sont facilement redimensionnées par un simple changement de configuration, mais cela implique généralement des changements au niveau du matériel et du système de fichiers pour ajuster les partitions, ce qui complique le redimensionnement dynamique. Archiver permet d'éviter ces problèmes en utilisant un seul volume pour tout, avec pour compromis un ralentissement de la vitesse d'E/S.

Fichiers manifest

Cette rubrique décrit les fichiers manifest et en donne un exemple pour un fichier de métabase de données. Elle décrit également la recherche du fichier manifest et en fournit un exemple.

Les fichiers manifest sont créés avec chaque fichier de base de données de sessions, de métadonnées et de paquets (logs) et le répertoire des tranches d'index. Un fichier manifest est un fichier qui décrit plusieurs éléments d'information importants sur les données auxquelles il fait référence. Les fichiers manifest sont écrits sous la forme d'un enregistrement JSON. Ils circulent avec les données qu'ils représentent de niveau en niveau. Si les données qu'ils représentent sont supprimées, le fichier manifest est également supprimé, sauf dans le cas particulier suivant. Si le service dispose de `/database/config/manifest.dir` configuré dans un répertoire valide, au moment où les données du fichier manifest sont supprimées, une copie du fichier manifest est placée dans le répertoire pointé par `manifest.dir` (le répertoire est créé s'il n'existe pas). Cela active une fonctionnalité NetWitness Suite appelée recherche de l'historique des fichiers manifest.

Le but de ce processus est de conserver un historique des fichiers manifest sur des années, à un emplacement donné pour une interrogation hors ligne. Comme vous pouvez l'imaginer pour un service en cours d'exécution pendant de nombreuses années, cela peut éventuellement générer des centaines de milliers de fichiers. Cela ne devrait pas être un sujet de préoccupation, car le service compresse automatiquement les fichiers en une seule archive afin d'économiser de l'espace lorsqu'ils deviennent trop nombreux. Les fichiers manifest sont très petits et se compressent bien

Exemple de fichier manifest (`meta-000000023.nwmdb.manifest`) pour un fichier de base de données :

```
{
    "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
```

Dans cet exemple de fichier manifest, les domaines les plus importants sont `fileTime` , `time1` et `time2` . Les trois champs sont écrits au format de temps POSIX. `time1` et `time2` sont les heures de début et de fin de l'enregistrement des métadonnées dans le fichier de base de métadonnées `meta-000000023.nwmdb` . En particulier, `fileTime` désigne toujours l'heure à laquelle le fichier a été créé (et non la dernière modification). `time1` et `time2` représentent la plage de données avec sa valeur minimale et maximale de données analysées au sein du fichier de base de métdonnées. Lorsque vous effectuez des recherches dans l'historique en fonction d'une période donnée, les plages `time1` et `time2` sont préférées à `fileTime` , si elles sont disponibles. Les fichiers manifest des autres bases de données et de l'index contiennent différents champs, mais tous contiennent suffisamment d'informations pour effectuer des requêtes basées sur le temps.

Effectuer une recherche dans l'historique des fichiers manifest

Lorsque les manifest sont collectés dans le répertoire pointé par `manifest.dir` , on suppose que les données de référence ont été copiées au niveau Cold et éventuellement sauvegardées dans le stockage hors ligne. Parce que l'historique des fichiers manifest est toujours accessible par le service, il est possible d'effectuer des requêtes sur une période sur les données hors ligne, afin de déterminer quelles données doivent être restaurées pour un intervalle de temps donné.

Vous pouvez rechercher des manifest à l'aide de la commande `/database manifest` :

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

Exemple de recherche :

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

La recherche renvoie tous les manifest qui correspondent à la requête :

```
[ 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 ]
```

Les résultats renvoyés peuvent être utilisés pour corréler les fichiers devant être restaurés à partir de la sauvegarde pour l'intervalle de temps donné. Pour NetWitness Suite 10.4 et version ultérieure, un service appelé Workbench peut être utilisé pour récupérer les fichiers restaurés et fournir une interface de requête sur les données restaurées à l'aide d'une ou de plusieurs collections.

La configuration de ces services n’entre pas dans le cadre du présent document. Pour plus d’informations, reportez-vous à la section « Configurer la sauvegarde et la restauration de données » dans le *Guide de configuration d’Archiver* .

Configuration avancée de la base de données

Cette section explique les options de configuration avancée de la base de données NetWitness Core.

Les options de configuration de la base de données NetWitness Core peut varier d'une version à l'autre. Cependant, de nombreux éléments de configuration ne changent pas fréquemment et sont expliqués ici. La liste n'est pas exhaustive, car de nouvelles fonctionnalités sont ajoutées à chaque version et peuvent nécessiter de nouveaux éléments de configuration. Pour consulter la documentation la plus récente, reportez-vous à la fonction d'aide intégrée dans le service NetWitness Core.

Rubriques

Nœuds de configuration de la base de données

Cette rubrique décrit les nœuds de configuration de la base de données. Les nœuds de configuration de la base de données sont des éléments de configuration de base de données avancés de la base de données NetWitness Core qui ne changent pas fréquemment.

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

Il s'agit de l'entrée de configuration primaire pour chaque base de données (également connue comme niveau Hot). Elle contrôle l'endroit du système de fichiers où les bases de données respectives sont stockées. Cette entrée de configuration appréhende une syntaxe complexe pour spécifier un nombre important de répertoires comme les emplacements de stockage.

Syntaxe de configuration :

```
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 ? ;
```

Exemple :

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

Les valeurs de taille sont facultatives : Si elles sont définies, elles indiquent la taille totale maximum des fichiers stockés à cet endroit avant le déploiement des bases de données. Si la taille n'est pas définie, la base de données n'est pas déployée automatiquement mais sa taille peut être gérée à l'aide d'autres mécanismes.

L'utilisation de = ou == est importante. Le comportement par défaut des bases de données consiste à créer automatiquement des répertoires lorsque le service Core démarre. Cependant, ce comportement peut être modifié à l'aide de la syntaxe == . En cas d'utilisation de == , le service ne crée pas de répertoires. Si les répertoires n'existent pas lorsque le service démarre, celui-ci ne peut pas commencer le traitement. Cela apporte de la résilience au service face à des systèmes de fichiers manquants ou non montés lorsque l'hôte démarre.

Si vous modifiez la taille d'un répertoire en cours d'utilisation, la modification prend effet immédiatement dans le cas où la taille choisie est supérieure. Si la taille est inférieure, elle est ignorée si elle plus de 10 % inférieure à la taille actuelle. Cela permet d'éviter les erreurs de frappe qui engendreraient une perte énorme de données. Par exemple, si la base de données de paquets est configurée pour `12 TB` et qu'une erreur de frappe indique `12 GB` , la base de données devrait supprimer plus de 11 To de données pour arriver à 12 Go. En pratique, la base de données ignore le paramètre de `12 GB` et consigne un avertissement afin que l'erreur soit repérée rapidement. Ainsi, si la taille indiquée est correcte et varie de plus de 10 % par rapport à la taille existante, la seule façon de la prendre en compte est de redémarrer le service. Lorsqu'il redémarre, le service assume que la taille est correcte et ajuste la base de données à la nouvelle taille en déployant les données les plus anciennes jusqu'à ce que la nouvelle taille soit atteinte. Si vous souhaitez ajuster la taille à la baisse de plus de 10 % sans redémarrer le service, vous devez modifier la taille plusieurs fois, de moins de 10 % chaque fois. Consultez les logs de service pour savoir quand la base de données s'est ajustée à la nouvelle taille, car la taille totale de la base de données ne s'ajuste que lorsque le dernier ficher écrit a été fermé.

Si de nouveaux répertoires sont ajoutés ou supprimés (séparés par des points-virgules), la modification n'est effective que lorsque le service redémarre.

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

Ces paramètres sont facultatifs et sont utilisés pour le niveau de stockage Warm dans Archiver. Par défaut, ils sont vides et inutilisés. S'ils sont configurés, ils suivent le même format et comportement que `packet.dir` , `meta.dir` et `session.dir` (reportez-vous à la section _ `packet.dir` , `meta.dir` et `session.dir` _ ci-dessus). Lorsqu'ils sont configurés, le fichier le plus ancien du niveau Hot passe au niveau Warm lorsqu'il n'y a plus d'espace disponible dans le niveau Hot.

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

Ces paramètres sont facultatifs et sont utilisés pour déplacer les fichiers du niveau de système de stockage Hot ou Warm vers le répertoire de niveau Cold spécifié. En fait, ce paramètre n'est autre qu'un répertoire, sans spécificateur de taille. Toutefois, le nom du chemin défini dispose de quelques spécificateurs de format que vous pouvez utiliser pour nommer le répertoire en y incluant la date des données.

```
%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
```

Paramètre fourni à titre d'exemple :

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

Pour le paramètre ci-dessus, si un fichier log de base de données est sur le point d'être déplacé vers le stockage à froid et qu'il a été créé le `2014-03-02 15:00:00` , il sera déplacé vers le répertoire suivant de niveau Cold :

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

Expliquons le dernier nombre `01` . Le spécificateur `%8r` sépare les heures de la journée en 24 / 8 = 3 parties. Les huit premières heures de la journée sont en mode bloc `00` , soit 12 h 00 à 8 h 00. Les huit heures suivantes vont de 8 h 00 à 16 h 00 et sont attribuées en mode bloc `01` . Les données déplacées vers le stockage à froid ont été créées à 15h00. Il s'agit donc du bloc `01` . Le spécificateur de format `%r` est utile pour sauvegarder les fichiers avec une granularité comprise entre un jour `%d` et une heure `%h` . Le répertoire de stockage Cold est créé à la demande et défini par le déplacement des données lorsque les spécificateurs de format sont utilisés.

La possibilité d'ajouter une date au chemin de données est simplement une commodité ajoutée pour la sauvegarde et la restauration. C'est une façon de baliser les données à l'aide d'une date comprise dans le chemin.

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

Permet de contrôler la taille des fichiers créés dans chaque base de données. Généralement, il n'est pas nécessaire de modifier ces valeurs car les valeurs par défaut fonctionnent bien. Ce paramètre prend effet immédiatement pour les fichiers qui suivent.

packet.files,meta.files,session.files

Ce paramètre contrôle le nombre de fichiers que la base de données conserve ouverts. Vous pouvez augmenter cette valeur pour améliorer les performances. Toutefois, le système d'exploitation impose une limite générale dans le nombre de fichiers que le service peut conserver ouverts. Si cette limite est dépassée, une erreur est signalée et le service ne fonctionne pas. Ce paramètre prend effet immédiatement.

Dans NetWitness Suite 10.6 et versions ultérieures, la valeur par défaut pour les packet.files, meta.files et session.files est `auto` et le service gère le nombre de fichiers ouverts en fonction des critères suivants :

1. Nombre de collections
2. Quantité de mémoire système

Lorsque le nombre est défini sur `auto` , il est dynamique et il s'affiche dans les logs lorsqu'il change. Pour NetWitness Suite 10.6, RSA recommande de définir cette valeur sur `auto` et de ne pas la remplacer par un nombre spécifique.

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

Ce paramètre fournit une limite de sécurité en termes d'espace disponible minimum sur les chemins spécifiés par les répertoires packet.dir, meta.dir et session.dir, respectivement. Ce paramètre est utilisé pour empêcher le service de manquer d'espace dans l'éventualité où d'autres programmes occuperaient l'espace réservé à chacune des bases de données. Ce paramètre prend effet immédiatement.

packet.index.fidelity,meta.index.fidelity

Ce paramètre contrôle la fréquence à laquelle les emplacements d'ID de paquets et les emplacements d'ID de méta sont indexés. Ce paramètre peut être augmenté pour réduire la quantité d'espace nécessaire à chaque fichier nwindex de paquet ou de méta, mais l'augmentation de ce paramètre réduit la vitesse à laquelle chaque paquet individuel ou élément méta peut être localisé. Ce paramètre prend effet immédiatement.

La base de données de sessions ne dispose pas de paramètre de fidélité car elle ne génère pas de fichiers d'index.

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

Ce paramètre contrôle si la base de données force un fonctionnement synchronisé sur le système de fichiers lorsque l'écriture d'un fichier est terminée. La valeur par défaut est `sync` , ce qui signifie que lorsqu'un fichier est fermé, il y aura un délai significatif pour que les données soient écrites sur un stockage non volatile. Il peut être nécessaire de définir cette valeur sur `normal` afin que le taux d'écriture reste supérieur, en particulier sur un Decoder. Ce paramètre prend effet à la création de fichier suivante. Ainsi, l'on peut s'attendre à une synchronisation de plus si la valeur vient de passer à `normal` .

Si des rejets de paquets se produisent et que la valeur packet.integrity.flush est définie sur `sync` , définissez-la sur `normal` et surveillez. Conservez les paramètres de la session et de vidage des méta sur `sync` . Si le problème de rejet des paquets persiste, définissez les trois valeurs sur `normal` et surveillez.

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

La taille de bloc représente la quantité de données allouée à la fois au sein de chaque fichier de base de données. Des blocs de plus grande taille peuvent proposer des débits et taux de compression plus élevés, et peuvent améliorer la vitesse à laquelle les éléments sont récupérés de la base de données sous forme séquentielle. Cependant, des blocs de grande taille ont un effet négatif sur la vitesse de lecture aléatoire des éléments de paquets et méta compressés. Ce paramètre prend effet immédiatement.

packet.compression,meta.compression

Ces paramètres contrôlent si les bases de données compressent les données. La compression réduit la quantité de stockage nécessaire pour chaque base de données, mais elle peut avoir un effet négatif majeur sur la vitesse à laquelle les éléments sont écrits dans la base de données, et la vitesse à laquelle les éléments sont récupérés de la base de données. Les modifications prennent effet immédiatement à la prochaine création de fichier.

À compter de Suite de NetWitness Suite 10.4, les valeurs valides pour ce paramètre sont `gzip` , `bzip2` , `lzma` ou `none` . `gzip` est l'algorithme utilisé de préférence avec la compression car il propose un bon équilibre entre performances et économies d'espace. `bzip2` et `lzma` permettent de meilleures économies d'espace, mais l'impact sur la vitesse est significatif et ils ne devraient être utilisés que pour les faibles vitesses de réception, et lorsque l'espace de stockage est précieux.

packet.compression.level,meta.compression.level

Vous pouvez utiliser ces paramètres pour affiner encore le comportement des algorithmes de compression. Ils n'ont aucun effet lorsque la compression est désactivée. Les valeurs valides se situent entre 0 et 9\. La valeur par défaut de zéro indique que le logiciel choisit le meilleur paramètre pour la vitesse et la compression. Les valeurs entre 1 et 9 sont utilisées comme échelle à ajuster entre performances (1) et compression (9). La valeur 9 donne généralement la meilleure compression pour un algorithme donné, mais avec les plus mauvaises performances. Le meilleur paramètre se situe vers le milieu, ce qui correspond à la valeur choisie par 0.

hash.algorithm

Ce paramètre contrôle le hachage des fichiers de base de données. La valeur par défaut est `none` , c'est-à-dire qu'aucun hachage n'est effectué. Les valeurs valides sont `none` , `sha256` , `sha1` ou `md5` . Les fichiers de base de données peuvent être hachés pour fournir la preuve qu'ils n'ont pas été manipulés depuis leur fermeture. Le hachage prend du temps et lorsqu'il est activé, il a un impact sur les performances de réception. Cette modification prend effet immédiatement.

hash.databases

Ce paramètre sélectionne les bases de données qui seront hachées. Les valeurs valides sont `session` , `meta` et `packet` , et elles sont séparées par une virgule lors du hachage de plusieurs bases de données. Cette modification prend effet immédiatement.

hash.dir

Ce paramètre est généralement vide, ce qui signifie que le fichier de hachage est créé dans le même répertoire que le fichier de base de données qui a été haché. Si ce paramètre est défini, le fichier de hachage est écrit dans le répertoire spécifié. Il peut être utilisé comme une forme de stockage à écriture unique pour la résilience contre la manipulation du hachage.

Les fichiers de hachage sont des fichiers XML de petite taille contenant le hachage encodé au format hex, ainsi que les métadonnées sur le fichier de base de données qui a été haché.

Nœuds de configurations d'index

Cette section décrit les nœuds de configuration d'index. Les nœuds de configuration de l'index sont des éléments de configuration de base de données avancés de la base de données NetWitness Core qui ne changent pas fréquemment.

index.dir

Le paramètre `index.dir` contrôle l'endroit auquel sont stockés les fichiers utilisés par l'index. Ce paramètre prend en charge la même syntaxe que les paramètres `packet.dir` , `meta.dir` et `session.dir` .

index.dir.warm

Stockage de niveau Actif pour les tranches d'index. Ce paramètre prend en charge la même syntaxe que les paramètres `packet.dir.warm` , `meta.dir.warm` et `ession.dir.warm` .

index.dir.cold

Stockage de niveau Inactif pour les tranches d'index. Ce paramètre prend en charge la même syntaxe que les paramètres `packet.dir.cold` , `meta.dir.cold` et `session.dir.cold` .

index.slices.open

Ce paramètre contrôle le nombre de tranches d'index dont l'ouverture est maintenue par l'index. Les requêtes ouvrent automatiquement les tranches d'index selon le besoin. Lorsque les requêtes se terminent, le moteur d'index peut maintenir les tranches ouvertes, afin que les requêtes suivantes s'exécutent plus rapidement. Les tranches les plus récentes seront maintenues ouvertes, car elles sont les plus susceptibles d'être utilisées par les requêtes.

Si les requêtes sur l'index requièrent que ce dernier ouvre les tranches, elles s'exécuteront plus lentement que si les tranches étaient déjà ouvertes. Ainsi, ce paramètre doit être affiné afin que la plupart des requêtes exécutées sur l'index fonctionnent sur les tranches ouvertes. Cependant, chaque tranche d'index ouverte consomme des ressources, comme des gestions de fichier et de la mémoire. Si un nombre trop important de tranches d'index est ouvert, les performances globales du service peuvent en pâtir.

Vous devez définir ce paramètre de sorte que les tranches d'index ouvertes couvrent la plupart des durées dont la plupart des requêtes auront besoin. Par exemple, si la plupart des requêtes sont supérieures aux deux semaines passées, et si des tranches d'index sont créées toutes les 8 heures, il existe 14 jours x 3 tranches par jour, soit 42 tranches créées au cours des deux dernières semaines. Ainsi, vous pouvez définir index.slices.open sur 42 pour que seules les tranches susceptibles d'être utilisées soient maintenues ouvertes.

Si ce paramètre est défini sur 0, toutes les tranches sont maintenues ouvertes jusqu'à la prochaine sauvegarde d'index. Dans ce scénario, le seul élément limitant le nombre de tranches ouvertes dans le processus est le nombre de tranches dans l'index.

page.compression

Déconseillé. Les versions 9.8 à 10.2 de l'index NetWitness Core prenaient en charge deux algorithmes de compression d'index différents ; vous pouvez en choisir un grâce à ce paramètre. À partir de la version 10.3, la seule valeur recommandée est la valeur par défaut de `huffhybrid`.

save.session.count

Ce paramètre contrôle la fréquence à laquelle l'index est automatiquement enregistré lors de l'insertion de nouvelles sessions. Si la valeur de save.session.count est supérieure à 0, dès qu'un nombre supérieur à `save.session.count` sessions est ajouté à l'index, ce dernier s'enregistre automatiquement. Si `save.session.count` est défini sur 0, cette fonction est désactivée et l'index ne s'enregistrera pas automatiquement lors de l'ajout de nouvelles sessions à l'index.

`save.session.count` peut être utilisé pour implémenter un schéma d'enregistrement automatique basé sur le volume de données entrant dans l'index. Cette fonction est utile car elle permet à un système légèrement chargé de générer des points d'enregistrement moins fréquemment.

Pour plus d'informations sur la section des enregistrements d'index, consultez la section de ce guide relative aux Techniques d'optimisation .

reindex.enable

Ce paramètre contrôle le fonctionnement du réindexeur d'arrière-plan .

Nœuds de configuration SDK

Cette rubrique décrit les nœuds de configuration SDK ayant un impact sur la base de données. Il existe des éléments de configuration supplémentaires pour chaque service Core ayant un impact sur la base de données mais pas sur la façon dont la base de données stocke ou récupère les données. Ces paramètres se trouvent dans le dossier `/sdk/config`.

max.concurrent.queries

Ce paramètre contrôle le nombre d'opérations de requêtes autorisées simultanément dans la base de données. Autoriser un nombre plus important d'opérations de requêtes simultanées peut améliorer la réactivité pour plus d'utilisateurs, mais la charge de requêtes du service Core reste étroitement liée aux E/S, et une valeur max.concurrent.queries élevée peut avoir un effet négatif. La valeur recommandée s'approche du nombre de cœurs du système, y compris dans le cas d'hyper-threading. Ainsi, pour une appliance avec 16 cœurs, la valeur doit se rapprocher de 32\. Réduisez-la un peu pour les threads d'agrégation et les threads de réponse du système général. Réduisez-la encore si le système est hybride (par exemple, un Decoder et un Concentrator s'exécutent sur la même appliance). Il n'y a pas de chiffre miracle, mais une valeur située entre 16 et 32 devrait bien fonctionner.

max.pending.queries

Ce paramètre contrôle la taille du backlog pour le moteur de requêtes de la base de données. Des valeurs élevées permettent à la base de données de mettre plus d'opérations en file d'attente d'exécution. Une requête en file d'attente ne progresse pas dans son exécution. Par conséquent, il peut être plus utile que le système produise des erreurs lorsque la file d'attente est complète, plutôt que de laisser sa taille devenir excessive. Toutefois, dans un système réalisant principalement des opérations en lot comme des rapports, une file d'attente volumineuse ne présente pas d'effet négatif.

cache.window.minutes

Ce paramètre contrôle une fonction du moteur de requêtes conçue pour améliorer la réactivité des requêtes lorsqu'il existe de nombreux utilisateurs simultanés. Pour plus d'informations sur la période de cache, consultez la section Techniques d'optimisation .

max.where.clause.cache

Le cache de clause Where contrôle la taille de la mémoire pouvant être consommée par les opérations de requêtes qui doivent produire un ensemble important de données temporaires pour évaluer le tri ou la comptabilisation. Si la taille du cache de clause Where présente un dépassement de capacité, la requête fonctionne quand même, mais elle est beaucoup plus lente. Si le cache de clause Where est trop volumineux, les requêtes peuvent allouer tellement de mémoire que le service est obligé de permuter ou manque de mémoire. Ainsi, la valeur multipliée par max.concurrent.queries doit toujours être bien inférieure à la taille de la RAM physique. Ce paramètre comprend les tailles sous la forme d'un nombre suivi d'une unité, par exemple `1.5 GB` .

max.unique.values

Les valeurs uniques maximales limitent la quantité de mémoire pouvant être consommée par la fonction Valeurs SDK. Les valeurs SDK génèrent une liste de valeurs uniques. Afin de produire des résultats précis, il peut être nécessaire de fusionner un grand nombre de valeurs uniques à partir de plusieurs tranches. Cet ensemble fusionné de valeurs doit être conservé en mémoire, ce paramètre existe donc pour limiter la quantité de mémoire que le jeu de valeurs fusionné peut consommer. La valeur par défaut limitera l'utilisation de la mémoire à environ 1/10ème de la RAM totale.

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

Ces paramètres sont disponibles dans NetWitness Suite 10.4 et versions antérieures.

Dans NetWitness Suite 10.4 et versions antérieures, la base de données Core prend en charge trois niveaux de priorité des requêtes. Un niveau de priorité est attribué à chaque utilisateur. Ainsi, jusqu'à trois groupes d'utilisateurs peuvent être définis pour améliorer les performances. Ces paramètres contrôlent le temps que chaque utilisateur est autorisé à exécuter les requêtes. Par exemple, cette valeur sera moins élevée pour les utilisateurs disposant de privilèges inférieurs et ils ne pourront pas utiliser toutes les ressources du service Core avec des requêtes de longue durée.

query.timeout

Ce paramètre est disponible dans NetWitness Suite 10.5 et versions supérieures.

Les niveaux de requête ont été remplacés dans NetWitness Suite 10.5 et versions ultérieures par l'expiration du délai des requêtes des comptes utilisateur. Pour les connexions approuvées, ces délais d'expiration sont configurés sur le serveur NetWitness Suite. Pour les comptes sur les services Core, il existe un nouveau nœud de configuration sous chaque compte appelé `query.timeout` . C'est le nombre maximum de minutes d'exécution de chaque requête. Lorsque cette valeur est définie sur zéro, le service Core n'applique aucune expiration du délai des requêtes.

max.where.clause.sessions

Ce paramètre est disponible dans NetWitness Suite 10.5 et versions supérieures.

Ce paramètre impose une limite au nombre de sessions qui peuvent être scannées par une requête unique. Par exemple, si un utilisateur sélectionne toutes les méta de sa base de données, la base de données arrête de traiter les résultats une fois que le nombre de sessions lues pour cette requête atteint cette valeur de configuration. La valeur 0 désactive cette limite.

Le nombre de sessions nécessaires au traitement complet d'une requête est égal au nombre de sessions qui correspondent à la clause WHERE de la requête, en supposant que tous les termes de la clause Where disposent d'un index adapté. S'il existe des termes non indexés dans la clause Where, la base de données doit lire plus de sessions et de méta, et atteint cette limite plus vite.

max.query.groups

Ce paramètre est disponible dans NetWitness Suite 10.5 et versions supérieures.

Ce paramètre impose une limite au nombre de groupes uniques collectés dans une requête unique. Par exemple, si une requête présente un groupe par clause avec plusieurs méta ayant un nombre élevé de valeurs uniques, la quantité de mémoire nécessaire à cette requête pourrait facilement dépasser la quantité de RAM disponible sur le serveur. Ainsi, cette limite existe pour éviter les insuffisances de mémoire.

La valeur 0 désactive cette limite.

packet.read.throttle

Ce paramètre réservé au Decoder a un impact sur l'accès à la base de données de paquets. Lorsque packet.read.throttle est défini sur une valeur supérieure à 0, le Décoder tente de réguler la lecture de paquets lorsqu'il détecte un conflit d'accès des paquets sur la base de données de paquets. Les nombres élevés proposent une régulation plus forte. Les modifications prennent effet immédiatement.

cache.dir,cache.size

Tous les services NetWitness Suite Core conservent un petit cache de fichiers de contenu brut extrait du périphérique. Ces paramètres contrôlent l'emplacement (cache.dir) et la taille (cache.size) de ce cache.

parallel.values

Ce paramètre est disponible dans NetWitness Suite 10.5 et versions supérieures.

Ce paramètre permet à des opérations SDK de s'exécuter en parallèle. S'il est défini sur 0, l'exécution parallèle est désactivée. S'il est défini sur une valeur supérieure à 0, il représente le nombre de threads créées lorsque chaque opération SDK s'exécute. La valeur maximale est le nombre de CPU logiques disponibles au début du processus.

Il est utile de définir une valeur élevée pour le paramètre parallel.values dans le cas où le nombre d'utilisateurs simultanés est faible, car cela permet d'exécuter plus rapidement des procédures d'enquête plus complexes. Si le nombre d'utilisateurs simultanés est élevé, il est préférable d'utiliser ici une valeur faible car il y aura un grand nombre d'opérations SDK indépendantes exécutées en même temps.

parallel.query

Ce paramètre est disponible dans NetWitness Suite 10.5 et versions supérieures.

Cette configuration est similaire au paramètre parallel.values en ce sens que la valeur maximum est le nombre de CPU logiques. La définition du paramètre parallel.query sur une valeur spécifique doit prendre en compte le nombre d'utilisateurs simultanés afin d'optimiser l'utilisation de la CPU sans dépasser régulièrement les ressources disponibles.

Il est utile de définir une valeur élevée pour le paramètre parallel.query dans le cas où le nombre de requêtes et d'utilisateurs simultanés est faible, car cela permet d'exécuter plus rapidement des requêtes plus complexes. Si le nombre de requêtes et d'utilisateurs simultanés est élevé, il est préférable d'utiliser une valeur faible car le nombre d'opérations de requêtes SDK indépendantes exécutées simultanément sera élevé.

Les opérations de requêtes sont limitées par le taux de lecture de la base de données méta. Par conséquent, la définition du paramètre parallel.query sur une valeur supérieure à 4 aura peu de chances de produire des résultats significativement supérieurs à la valeur par défaut de 0\. La valeur parallel.query la mieux adaptée dépendra du type de stockage connecté. Essayez différentes valeurs pour parallel.query afin de déterminer les meilleurs résultats avec votre système de stockage.

Nœuds de configuration par utilisateur

Cette rubrique décrit les nœuds de configuration par utilisateur. Il existe des paramètres qui influent sur les actions que les utilisateurs sont autorisés à effectuer sur la base de données. Ces paramètres sont stockés dans l'arborescence de configuration dans `/users/accounts/<username>/config` , où `<username>` est le nom de l'utilisateur auquel les paramètres s'appliquent.

query.prefix

Un préfixe de requête applique un filtre à chaque opération de requête que l'utilisateur exécute. Il est mis en œuvre en prenant les valeurs `query.prefix` et en les ajoutant à la clause where de chaque requête utilisant l'opérateur logique && (et). Pour plus d'informations sur les clauses Where, reportez-vous à la section Requêtes .

query.level

Ce paramètre est disponible dans NetWitness Suite 10.4 et versions antérieures.

Le paramètre query.level attribue le niveau de requête dont les utilisateurs disposent pour chaque requête qu'ils effectuent. Il a une influence si leurs requêtes sont limitées par les query.level.1.minutes, query.level.2.minutes ou query.level.3.minutes.

query.timeout

Ce paramètre est disponible dans NetWitness Suite 10.5 et versions ultérieures.

Le paramètre query.timeout attribue le montant maximum de temps en minutes pour l'exécution de chaque requête. Pour les connexions approuvées, ces délais d'expiration sont configurés sur le serveur NetWitness Suite. Pour les comptes des services Core, ce paramètre est stocké dans l'arborescence de configuration dans `/users/accounts/<username>/config` où `<username>` est le nom de l'utilisateur auquel les paramètres s'appliquent. Lorsque cette valeur est définie sur zéro, le service Core n'impose pas d'expiration du délai de requête.

session.threshold

Le paramètre session.threshold attribue un seuil maximal de session pour l'utilisateur. Si elle est définie, cette valeur de seuil est attribuée à tous les appels de valeurs que l'utilisateur effectue. Une présentation détaillée de l'appel des valeurs et des seuils est décrite dans ce guide.

Planificateur

Cette rubrique présente brièvement le planificateur et explique comment planifier des commandes. Tous les services NetWitness Core sont dotés d'un planificateur intégré disponible sous `/sys/config/scheduler` . Pour utiliser ce planificateur, vous devez ajouter la commande à exécuter périodiquement à l'aide de l'un des deux messages suivants :

`/sys/config/scheduler addInter` - Ajoute une commande à exécuter à l'intervalle spécifié (toutes les N heures, minutes ou secondes)

ou

`/sys/config/scheduler addMil` - Ajoute une commande à exécuter à l'heure du jour indiquée ou même à des jours spécifiques de la semaine

Exemple

Par exemple, supposez que vous ayez à supprimer toutes les données de paquet collectées pendant plus de sept jours. Comme vous ne pouvez pas configurer le paramètre packet.dir pour déployer les données à l'intervalle souhaité, vous devez planifier la commande `/database timeRoll` pour qu'elle soit exécutée au même intervalle. Pour cet exemple, créez une commande timeRoll qui sera exécutée toutes les 20 minutes :

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

Cette commande ajoute une tâche planifiée (maintenue entre les redémarrages du service) qui sera exécutée toutes les 20 minutes sur le nœud `/database` et rejette toutes les données de paquet de plus de sept jours. Le paramètre `params` permet de transférer tous les paramètres à la commande spécifiée (dans ce cas `timeRoll`). Notez l'ajout de guillemets autour des paramètres intégrés (`type` et `days`) pour que ceux-ci soient interprétés en tant que paramètres à transmettre à la commande sortante `addIter` . Si les paramètres de `params` comprennent des guillemets, vous devez neutraliser les guillemets intérieurs par une barre oblique inverse. Vous pouvez les réécrire avec des guillemets intégrés sans que cela est d'incidence sur la commande :

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

Cette commande fonctionne comme la commande d'origine, mais explique comment neutraliser un transfert de paramètres compliqué. Autres commandes utiles du planificateur :

`/sys/config/scheduler print` - Imprime toutes les commandes planifiées (vous pouvez également les afficher en exécutant une commande `ls` sur le nœud du planificateur).

`/sys/config/scheduler delSched` - Supprime une commande planifiée en passant l'identifiant affiché dans la commande `print` (ou `ls`).

Brève présentation du planificateur. Pour plus d'informations sur les paramètres de commande, envoyez le message `help` au nœud du planificateur et transmettez le nom de la commande via les paramètres `msg` . Pour plus d'informations, reportez-vous à la rubrique « Vue Explorer les services » du *Guide de mise en route de l'hôte et des services* .

RollOver

Cette rubrique décrit les deux mécanismes de transfert. La base de données fonctionne sur la base du premier entré, premier sorti (FIFO, first-in, first-out). De nouvelles données sont toujours ajoutées à la base de données, et les données les plus anciennes sont supprimées automatiquement au besoin. Les données qui se trouvent au milieu de la base de données sont immuables, ce qui signifie qu'elles ne peuvent pas être modifiées.

Il existe deux mécanismes de transfert : synchrone et asynchrone.

Transfert synchrone

Le transfert synchrone fait référence aux paramètres de transfert qui sont appliqués en réponse à une opération d'écriture sur la base de données. Cela signifie que les données sont supprimées de la base de données en réponse directe à la nécessité d'écrire de nouvelles données. Le transfert synchrone est configuré en définissant les valeurs de taille de la configuration de `packet.dir` , `meta.dir` , `session.dir` et `index.dir` .

Le transfert synchrone sur les bases de données de paquets, de métadonnées et de sessions peut s'effectuer dans n'importe quelle opération d'écriture. Le transfert synchrone sur l'index se produit lorsque ce dernier est enregistré.

Transfert asynchrone

Le transfert asynchrone fait référence à la suppression de fichiers de la base de données qui se produit lorsqu'une commande de transfert explicite est envoyée à la base de données. Le plus souvent, ce type de transfert est planifié pour s'exécuter périodiquement à l'aide du planificateur intégré du service Core. L'utilisateur peut également le demander explicitement.

La commande de transfert asynchrone est le message `sizeRoll` sur les nœuds `/index` et `/database` de l'arborescence de configuration. Le message du nœud `/database` dimensionne réellement le transfert sur les bases de données de paquets, de métadonnées et de sessions uniquement, alors que message du nœud `/index` peut effectuer le transfert simultané à la fois sur l'index et les bases de données de paquets, de métadonnées et de sessions.

La commande `sizeRoll` a la syntaxe de paramètre suivante :

```
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 ? ;
```

Le paramètre `type` contrôle les bases de données à prendre en compte pour la suppression des données les plus anciennes en fonction de la taille totale ou l'espace restant. Si le type n'est pas spécifié pour la commande `/index` sizeRoll, seul l'index est pris en compte pour les opérations de transfert.

Le paramètre `maxSize` définit la taille maximale actuelle de la base de données ou de l'index. Si la base de données est supérieure à cette taille, les données les plus anciennes seront supprimées en premier (ou déplacées vers le niveau Warm ou Cold, en fonction de la configuration) jusqu'à ce que la taille totale soit inférieure à `maxSize` . L'opération `sizeRoll` détermine les données les plus anciennes sur toutes les bases de données et l'index en fonction des ID de session. Les entrées de session ou d'index avec les ID de session les plus bas sont supprimés en premier, y compris éventuellement la suppression des bases de métadonnées et des bases de données de paquets qui sont orphelines en supprimant les entrées de la base de données de session. Les données d'index sont transférées si les sessions auxquelles elles font référence sont supprimées.

Le paramètre `maxSizeWarm` définit une taille maximale actuelle au niveau *Warm* , mais autrement se comporte de la même manière que le paramètre `maxSize` . Lorsque les données sont transférées au niveau Warm, elles sont déplacées vers le niveau Cold (si configuré) ou supprimées.

Le paramètre `maxPercent` définit un pourcentage maximal de tous les volumes de toutes les bases de données dans le paramètre `type` combiné. Une fois la limite dépassée, les données les plus anciennes sont supprimées en premier jusqu'à ce que la taille totale soit inférieure à la valeur maxPercent des volumes totaux.

Le paramètre `minFree` définit un espace libre minimum autorisé sur les volumes avant que les données les plus anciennes soient supprimées.

Chaque appel à l'opération sizeRoll fournit une voie d'accès unique à la base de données pour supprimer les fichiers. Une fois l'opération terminée, l'utilisation de la taille actuelle de la base de données aura satisfait aux critères spécifiés par les paramètres `maxSize` , `maxPercent` ou `minFree` et le paramètre `maxSizeWarm` facultatif. Par conséquent, cette opération peut être programmée périodiquement pour s'assurer que la base de données peut continuer à fonctionner sans interruption.

Exemple

L'exemple suivant montre une entrée classique du planificateur sizeRoll pour un Archiver :

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

Cette entrée de planificateur indique que toutes les cinq minutes, la base de données vérifie que la taille maximale des métadonnées, des sessions, des paquets et de l'index ne dépasse pas 25 téraoctets sur le niveau Hot et ne dépasse pas 150 téraoctets sur le niveau Warm.

Requêtes

Cette rubrique couvre la syntaxe de requête de la base de données. Il existe trois principaux mécanismes pour effectuer des requêtes dans la base de données : les appels de `query` , `values` et `msearch` dans le dossier `/sdk` pour chaque service Core.

L'appel de `query` renvoie des éléments méta depuis la base de métadonnées, éventuellement à l'aide de l'index pour une récupération rapide.

L'appel de `values` renvoie des groupes de métavaleurs uniques triés selon certains critères. Il est optimisé pour renvoyer un sous-ensemble des valeurs uniques triées selon une fonction d'agrégat, comme un nombre.

L'appel de `msearch` effectue la recherche sur le texte saisi et renvoie les sessions qui correspondent aux termes recherchés. Il peut rechercher dans les index, les méta, les paquets bruts ou les logs bruts.

`query` Syntaxe

Le message de `query` possède la syntaxe suivante :

```
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 ? ;
```

Les paramètres `id1` , `id2` et `size` forment un mécanisme de pagination pour renvoyer un grand nombre de résultats à partir de la base de données. Leur utilisation est principalement utile pour les développeurs, qui écrivent des applications directement par rapport à la base de données NetWitness Core. Généralement, les résultats sont renvoyés dans l'ordre des données les plus anciennes aux plus récentes (les ID de métadonnées supérieurs sont toujours plus récents). Pour pouvoir renvoyer des résultats des plus récents aux plus anciens, inversez les ID de sorte que `id1` soit supérieur à `id2` . Les performances en sont légèrement pénalisées, car la clause where doit être entièrement évaluée avant que le traitement en ordre inversé puisse commencer.

Lorsque la taille est ignorée ou définie sur zéro, le système renvoie tous les résultats sans pagination. Dans l'interface RESTful, cela cause le renvoi de la réponse complète avec un codage par fragments. Le protocole natif renvoie les résultats sur plusieurs messages.

Le paramètre `query` est une chaîne de commande de paramètres `query` possédant sa propre syntaxe spécifique NetWitness :

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

La clause `select` vous permet de spécifier `*` pour renvoyer toutes les métadonnées de toutes les sessions correspondant à la clause where, ou un ensemble de noms de champ de métadonnées et de fonctions agrégées pour sélectionner un sous-ensemble de métadonnées avec chaque session.

La clause `select` peut contenir des noms de clé méta renommées. Les champs qui figurent dans l'ensemble de résultats comme résultat d'une clé renommée dans la clause `select` sont renvoyés avec le nom de clé méta correspondant au nom utilisé dans la clause `select` . Par exemple, si la clé `port_src` est utilisée pour renommer `tcp.srcport` , une requête contenant `select port_src` retourne seulement les champs `port_src` , même si les métadonnées sous-jacentes avaient le type `tcp.srcport` .

Les fonctions d'agrégat exercent l'effet suivant sur l'ensemble de résultats de la requête.

Fonction	Résultat
`sum`	Somme de toutes les valeurs de métadonnées ; ne fonctionne qu'avec les nombres
`count`	Nombre total de champs de métadonnées qui auraient été renvoyés
`min`	Valeur minimale affichée
`max`	Valeur maximale affichée
`avg`	Valeur moyenne pour le nombre
`distinct`	Renvoie la liste de toutes les valeurs uniques affichées
`countdistinct`	Renvoie le nombre de valeurs uniques affichées. `countdistinct` est équivalent au nombre de métadonnées qui auraient été renvoyées par la

	fonction distinct.
`first`	Renvoie la première valeur affichée
`last`	Renvoie la dernière valeur affichée
`len`	Convertit toutes les valeurs de champ en longueur UInt32 plutôt que de retourner la valeur réelle. Cette longueur est le nombre d'octets permettant de stocker la valeur réelle, non la longueur de la structure stockée dans la métabase de données. Par exemple, le mot « NetWitness » renvoie une longueur de 10. Tous les champs IPv4, comme `ip.src` , renvoient 4 octets.

`where` Clauses

La clause `where` est une spécification de filtre qui vous permet de sélectionner des sessions dans la collection en utilisant l'index.

Syntaxe :

```
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" ;
```

Lorsque vous spécifiez des critères de règle, la partie `meta-value` de la clause doit correspondre au type de métadonnées spécifié par `meta-key` . Par exemple, si la clé est `ip.src` , la `meta-value` doit être une adresse IPv4.

Les requêtes utilisant un nom `meta-key` correspondront à des éléments méta correspondant aux noms `meta-key` ainsi qu'aux noms de n'importe quel « changement de nom » spécifié pour la clé. Reportez-vous à la section « Changement de nom de clé » dans la rubrique Personnalisation d'index pour plus d'informations sur la modification du nom des clés.

Opérateurs de requête

Le tableau suivant décrit la fonction de chaque opérateur.

Opérateur	Fonction
=	Associez les sessions contenant exactement la valeur de métadonnées. Si la gamme de valeurs est spécifiée, toutes les valeurs sont considérées comme une correspondance.
!=	Associe toutes les sessions qui ne correspondraient pas à la clause same comme si elles étaient écrites avec l'opérateur = .
<	Pour les valeurs numériques, associe les sessions contenant des métadonnées dont la valeur numérique est inférieure au côté droit. Si le côté droit contient une gamme, sa première valeur est prise en compte. Si plusieurs gammes sont spécifiées, le comportement n'est pas défini. Pour les métadonnées texte, une comparaison lexicographique est effectuée.
<=	Même comportement que < , mais les sessions contenant une méta exactement égale à la valeur sont également considérées comme des correspondances.
>	Semblable à l'opérateur < , mais associe les sessions dans lesquelles la valeur numérique est supérieure au côté droit. Si le côté droit est une gamme, la dernière valeur de la gamme est prise en compte pour la comparaison.
>=	Même comportement que > , mais les sessions contenant une méta exactement égale à la valeur sont également considérées comme des correspondances.
`begins`	Associe des sessions qui contiennent une valeur de métadonnées texte qui commence par les mêmes caractères que le côté droit.

`ends`	Associe des sessions qui contiennent une valeur de métadonnées texte qui se termine par les mêmes caractères que le côté droit.
`contains`	Associe des sessions qui contiennent une valeur de métadonnées texte contenant la sous-chaîne indiquée sur le côté droit.
`regex`	Associe des sessions qui contiennent une valeur de métadonnées texte correspondant au regex donné sur le côté droit. L'analyse syntaxique de regex est gérée par boost::regex.
`exists`	Associe des sessions qui contiennent toutes les valeurs de métadonnées avec la clé méta donnée.
`!exists`	Associe des sessions qui ne contiennent pas toutes les valeurs de métadonnées avec la clé méta donnée.
`length`	Associe des sessions qui contiennent des valeurs de métadonnées texte d'une certaine longueur. L'expression sur le côté droit doit être un nombre non négatif.

Valeurs de texte

Le système attend des valeurs de texte entre guillemets. Sauf si elle peut être analysée en tant que durée (voir ci-dessous), une valeur entre guillemets est interprétée comme du texte.

Adresses IP

Les adresses IP peuvent être exprimées à l'aide de représentations de texte standard pour les adresses IPv4 et IPv6. De plus, la requête peut utiliser la notation CIDR pour exprimer une plage d'adresses. Si la notation CIDR est utilisée, elle est développée à la plage de valeur équivalente.

Adresses MAC

Une adresse MAC peut être spécifiée grâce à la notation d'adresse MAC standard : `aa:bb:cc:dd:ee:ff`

Expressions de date et d'heure

Dans NetWitness Suite, les dates sont représentées à l'aide de l'heure Epoch Unix, qui est le nombre de secondes depuis le 1er jan. 1970 UTC. Dans les requêtes, vous pouvez exprimer l'heure sous forme de ce nombre en secondes, ou vous pouvez utiliser la représentation de chaîne. La représentation de chaîne de la date et de l'heure est `"YYYY-mmm-DD HH:MM:SS"` . Une abréviation à trois lettres représente le mois. Vous pouvez également exprimer le mois sous forme de nombre à deux chiffres, de 01 à 12.

Les valeurs d'heure doivent être entre guillemets.

Toutes les heures spécifiées dans les requêtes doivent être en UTC.

Points de temps relatif

Les points de temps relatifs permettent à une clause where de référencer une valeur avec un décalage fixe, par rapport aux premières ou dernières métadonnées vues dans la collection.

Une expression de point de temps relatif a la syntaxe `rtp(boundary, duration)` .

La limite est `earliest` ou `latest` .

La durée est une expression d'heures, minutes ou secondes. Par exemple, `24h` , `60m` ou `60s` .

Les points de temps relatif peuvent être utilisés uniquement dans les opérations de SDK qui comportent une collection qui fournit les limites pour les premières et dernières métadonnées de temps.

Les points de temps relatif fonctionnent uniquement sur les types de métadonnées indexées. Les types de métadonnées indexées par défaut sont `time` et `event.time` .

Exemples :

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


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

Valeurs de gamme spéciales

Les gammes sont généralement exprimées avec la syntaxe * `smallest` * - * `largest` *, mais vous pouvez utiliser certaines valeurs spéciales d'espace réservé dans les expressions de gamme. Vous pouvez utiliser la lettre `l` pour représenter la limite inférieure de toutes les valeurs de métadonnées en tant que début de la gamme, et la lettre `u` pour représenter la limite supérieure. Les limites sont déterminées par l'observation de la métavaleur inférieure ou supérieure dans l'index, parmi toutes les métavaleurs qui ont déjà été saisies dans l'index.

Si vous utilisez la balise `l` ou `u` , elle ne doit pas être entre guillemets.

Par exemple, l'expression `time = "2014-may-20 11:57:00" - u` correspondrait à toutes les heures à partir de 2014-may-20 11:57:00 jusqu'à l'heure la plus récente trouvée dans la collection.

Notez qu'il est facile de confondre une expression de gamme avec une chaîne de texte. Vérifiez que toutes les valeurs de texte qui contiennent - sont entre guillemets et que les tirets dans des expressions de gamme ne sont pas entre guillemets.

Clause `group by` (à partir de 10.5)

L'API de requête peut générer des groupes agrégés à partir des résultats d'un appel de requête. Ceci s'effectue à l'aide d'une clause `group by` sur la requête. Lorsque `group by` est spécifié, l'ensemble de résultats pour la requête est sous-divisé en groupes. Chaque groupe de résultats est identifié de façon unique par les valeurs de métadonnées indiquées dans la clause group by.

Prenez par exemple la requête `select count(ip.dst)` . Cette requête renvoie le nombre de toutes les métadonnées ip.dst dans la base de données. Cependant, si vous ajoutez une clause `group by` , comme suit : `select count(ip.dst) group by ip.src` , la requête retourne le nombre de métadonnées ip.dst trouvées pour chaque ip.src unique.

À partir de NetWitness Suite version 10.5, vous pouvez utiliser jusqu'à 6 champs de métadonnées dans une clause `group by` .

La clause `group by` partage certaines des fonctionnalités de l'appel de `values` , mais offre des groupes beaucoup plus avancés, allongeant ainsi la durée des requêtes. La production des résultats d'une requête groupée implique la lecture de la métadonnée à partir de la base de métadonnées pour toutes les sessions qui correspondent à la clause `where` , tandis qu'un appel de valeurs peut produire ses agrégats en lisant uniquement l'index.

Le contenu de chaque groupe renvoyé par la requête est défini par la clause `select` . La clause `select` peut contenir l'une des fonctions agrégées ou des champs de métadonnées sélectionnés. Si plusieurs agrégats sont sélectionnés, le résultat de la fonction agrégée est défini pour chaque groupe. Si des champs non agrégés sont sélectionnés, les champs de métadonnées sont renvoyés par lots pour chaque groupe.

L'ensemble de résultats d'une requête `group by` est codé avec les règles suivantes :

1. Tous les éléments méta associés à un groupe sont fournis avec le même numéro de groupe.
2. Les premiers éléments méta renvoyés au groupe identifient la clé de groupe. Par exemple, si la clause `group by` spécifie `group by ip.src` , le premier élément méta de chaque groupe sera un `ip.src` .
3. Les éléments méta normaux et non agrégés sont renvoyés après la `group key` , mais ils auront toujours tous le même numéro de groupe que les métadonnées clés de groupe.
4. Les champs de métadonnées de résultat agrégé pour chaque groupe sont renvoyés ensuite.
5. Tous les champs dans un groupe sont renvoyés ensemble. Différents résultats de groupe seront entrelacés.

S'il manque l'un des éléments méta `group by` dans l'une des sessions associées par la clause `where` , ce champ de métadonnées est traité comme NULL pour ce groupe. Lorsque les résultats de ce groupe sont renvoyés, les parties à valeur NULL de la clé de groupe seront omises des résultats du groupe, car la base de données ne possède pas de concept NULL.

La sémantique d'une requête `group by` diffère d'une base de données de type SQL par rapport aux champs de métadonnées renvoyés. Les bases de données SQL nécessitent que vous sélectionniez de façon explicite les colonnes `group by` dans la clause `select` si vous souhaitez qu'elles soient renvoyées dans l'ensemble de résultats. La base de données NetWitness Core renvoie toujours de façon implicite les colonnes de groupe en premier.

Une requête avec une clause `group by` est conforme au paramètre `size` d'ensemble de résultats, s'il est fourni. Cependant, en raison de la nature du regroupement, l'appelant doit faire des efforts supplémentaires pour mettre en page et reformer les groupes si un ensemble de résultats à taille fixe est requis. C'est pour cette raison que vous ne devez pas spécifier une taille de résultat explicite lorsque vous faites un appel de `group by` . Si vous ne spécifiez pas de taille explicite, l'ensemble des résultats sera livré sous forme de résultats partiels.

Le tableau suivant décrit les paramètres de configuration de conformité de base de données qui limiteront l'E/S ou l'impact sur la mémoire d'un groupe par requête.

Paramètre	Fonction
`/sdk/config/max.query.groups`	Il s'agit de la limite du nombre de groupes pouvant être conservés en mémoire pour calculer des agrégats. Ce paramètre vous permet de limiter l'utilisation de la mémoire globale de la requête.
`/sdk/config/max.where.clause.sessions`	Il s'agit de la limite du nombre de sessions de la clause where pouvant être traitées dans une requête. Ce paramètre vous permet de définir une limite du nombre de sessions devant être lues à partir des métabases de données et des bases de données de session pour résoudre une requête.

Clause `order by` (à partir de 10.5)

Une clause `order by` peut être ajoutée à une requête qui contient une clause `group by` . La clause `order by` entraîne le renvoi de l'ensemble des résultats groupés dans l'ordre trié.

Une clause `order by` se compose d'un ensemble d'éléments à trier, dans l'ordre croissant ou décroissant. Le tri peut être effectué sur tout champ de données qui sera renvoyé dans un ensemble de résultats. Ceci inclut la métadonnée spécifiée par la clause `select` , les résultats de la fonction aggregate spécifiés par la clause `select` , ou les champs de métadonnées `group by` .

La clause `order by` peut faire le tri sur de nombreuses colonnes. Il n'existe pas de limite au nombre de colonnes `order by` autorisées dans la requête, mais il existe une limite pratique car chacune des colonnes `order by` peut se rapporter à un élément renvoyé par la clause `select` ou la clause `group by` . Le tri de colonne multiple est imposé de façon lexicographique, ce qui signifie que si deux groupes possèdent des valeurs égales pour la première colonne, ils sont ensuite triés selon la deuxième colonne. S'ils sont égaux dans la deuxième colonne, ils sont triés selon la troisième colonne, et ainsi de suite pour le nombre de colonnes `order by` fournies.

La base de données NetWitness Core est unique car les groupes de résultats renvoyés par une requête peuvent chacun avoir plusieurs valeurs pour une sélection. Par exemple, il est possible de sélectionner tous les éléments méta qui correspondent à un type de métadonnée et de les organiser en groupes, et il est possible d'utiliser la fonction `distinct()` pour renvoyer des groupes de métavaleurs distinctes. Si une clause `order by` fait référence à l'un des champs du groupe possédant plusieurs valeurs, l'ordre de tri est appliqué comme suit :

1. Dans chaque groupe, les champs possédant plusieurs valeurs de correspondance sont organisés selon la clause de classement
2. Tous les groupes sont triés par la comparaison de la première occurrence du champ ordonné trouvé dans chaque groupe

La clause `order by` n'est disponible que dans les requêtes qui possèdent une clause `group by` , car les groupes sont obligatoires pour organiser les champs de métadonnées en enregistrements distincts. Si vous souhaitez trier une requête arbitraire comme si aucun groupement n'était appliqué, utilisez le `group by sessionid` . Ceci permet de s'assurer que les résultats sont renvoyés en groupes de sessions ou événements distincts.

Les clauses `group by` sont naturellement renvoyées dans un ordre de clé de groupe croissant, mais une clause `order by` peut être utilisée pour renvoyer des groupes dans un ordre différent.

Si une colonne `order by` ne spécifie pas `asc` ni `desc` , l'ordre par défaut est croissant.

Exemples :

```
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
```

Appel de `values`

L'index fournit une fonction de bas niveau `values` pour accéder aux valeurs de métadonnées uniques qui ont été stockées dans l'index. Cette fonction permet aux développeurs d'effectuer des opérations plus avancées sur des groupes de valeurs de métadonnées uniques.

Syntaxe du paramètre d'appel `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 ;
```

L'appel de `values` fournit la fonction de renvoi d'un ensemble de valeurs de métadonnées uniques pour une clé méta donnée. Pour chaque valeur unique, l'appel de `values` peut fournir un nombre total agrégé. La fonction utilisée pour générer le total est contrôlée par le paramètre des balises.

Paramètres

Le tableau suivant décrit la fonction de chaque paramètre.

Paramètre	Fonction
`fieldName`	Il s'agit du nom de clé méta pour lequel vous récupérez des valeurs uniques. Par exemple, si `fieldName` est `ip.src` , cette fonction renvoie les valeurs IP source uniques dans la collection. Si le `fieldName` fait référence à une clé de références de changement de nom, le résultat est défini en tant que l'ensemble de valeurs de champ pour le nom de clé méta donnée ainsi que toutes méta clés des références à.
`where`	Il s'agit d'une clause `where` qui filtre l'ensemble de sessions pour lesquelles les valeurs uniques sont renvoyées. Par exemple, si le `fieldName` est `ip.src` et la clause `where` est `ip.src = 192.168.0.0/16` , uniquement dans la plage de valeurs `192.168.0.0` à `192.168.255.255` sont renvoyés. Pour plus d'informations sur la syntaxe de la clause `where` , voir *Clauses Where* .
`size`	Taille de l'ensemble de valeurs uniques à renvoyer. Cette fonction est optimisée pour renvoyer un petit sous-ensemble de valeurs uniques possibles dans la base de données.
`id1` , `id2`	Ces paramètres facultatifs limitent le périmètre de la recherche de valeurs uniques à une région spécifique de la base de métadonnées et de l'index. La définition des paramètres `id1` et `id2` sur une gamme limitée de la base de données méta est très importante pour exécuter des recherches sur de grandes collections.
`flags`	Les balises contrôlent la façon dont les valeurs sont triées et additionnées. Les balises sont décrites dans la section Balises de valeurs suivante.
`threshold`	La définition du paramètre `threshold` permet à l'appel de `values` d'ignorer la collection du total associé à chaque valeur une fois le seuil atteint. En fournissant un seuil, l'appelant peut réduire la quantité d'éléments d'index et de méta qui doivent être récupérés à partir de la base de données. Si le paramètre `threshold` est omis ou défini sur 0, cette optimisation n'est pas

	utilisée.
`aggregateFunction`	Paramètre facultatif utilisé pour modifier le comportement par défaut des sessions de comptage, des paquets ou de la taille pour le comptage ou la somme du champ numérique défini par `aggregateFieldName` . Les deux paramètres doivent être spécifiés lorsque l'un des deux est défini. Passez `sum` ou `count` pour spécifier le comportement à suivre.
`aggregateFieldName`	Le champ de métadonnées sur lequel exécuter le `aggregateFunction` . Les paramètres `aggregateFunction` et `aggregateFieldName` doivent être spécifiés lorsque la balise `aggregate` est définie. Un appel de `values` utilisant l'une des fonctions d'agrégat peut être beaucoup plus lent qu'un appel de `values` collectant des totaux de sessions, paquets ou taille. En effet, chaque session qui correspond à la clause `where` doit être récupérée à partir de la base de métadonnées. En raison de cette recherche, une grande partie de la requête est liée à l'E/S sur les volumes de la métabase de données. Le temps nécessaire à l'exécution d'un appel de `values` d'agrégat est linéairement proportionnel au nombre de sessions correspondant à la clause `where` .
`min` , `max`	Valeurs minimale et maximale qui doivent être renvoyées à partir de l'appel. Ces paramètres permettent d'effectuer une itération (ou une mise en page) sur un nombre de valeurs extrêmement important, généralement plusieurs valeurs qui pourraient être renvoyées à partir d'un appel unique. Principalement utilisé en association avec les balises `sort-value`,`sort-ascending` de sorte que la valeur supérieure renvoyée serait utilisée dans un appel suivant en tant que valeur de paramètre `min` . Les valeurs sont exclusives. Si `min="rsa"` était spécifié et `rsa` était une valeur valide, `rsa` ne serait pas renvoyé, mais la prochaine valeur supérieure serait renvoyée.

Balises `values`

Le paramètre `flags` contrôle le fonctionnement de l'appel de valeurs. Il existe trois groupes de balises qui correspondent aux différents modes de fonctionnement, tel que l'indique le tableau ci-dessous.

Balise	Description
`sessions` , `size` , `packets`	L'appel de `values` vous permet de spécifier l'une de ces balises pour déterminer la façon dont le total de chaque valeur est calculé. Si la balise est `sessions` , l'appel de `values` renvoie un nombre de sessions qui contiennent chaque valeur. Si la balise est `size` , l'appel de `values` établit le total de la taille de toutes les sessions qui contiennent chaque valeur unique, et établit un rapport de la taille totale de chaque valeur unique. Si la balise est `packets` , l'appel de valeur établit le total du nombre de paquets dans toutes les sessions qui contiennent chaque valeur unique, et établit un rapport de ce total pour chaque valeur unique.
`sort-total` , `sort-value`	Les balises contrôlent la façon dont les résultats sont triés. Si la balise est `sort-total` , l'ensemble de résultats est trié dans l'ordre des totaux collectés. Si la balise est `sort-value` , les résultats sont renvoyés dans l'ordre de tri des valeurs.
`order-ascending` , `order-descending`	Ces balises contrôlent l'ordre de tri de l'ensemble de résultats. Par exemple, en cas de tri par le total dans l'ordre décroissant, les valeurs possédant le total supérieur sont renvoyées en premier.

Exemple d’appel de `values`

L’appel de `values` est largement utilisé par la vue Navigation dans NetWitness Suite. La vue par défaut génère des appels ressemblant à celui-ci :

```
/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\""
```

Dans cet exemple, la vue Navigation requiert des valeurs uniques pour `ip.src` . Elle requiert des valeurs uniques d' `ip.src` pour la période donnée. Elle demande le nombre de sessions qui correspondent à chaque `ip.src` . Les résultats sont les 20 valeurs `ip.src` supérieures lorsqu'elles sont triées par le nombre total de sessions dans l'ordre décroissant. De plus, la vue Navigation possède une gamme d'ID de métadonnées afin de fournir un conseil d'optimisation au moteur de requête.

Appel de `msearch`

L'index propose une fonction `msearch` de faible niveau afin de réaliser des recherches de texte par rapport à tous les types de méta. Ce type de recherche ne requiert pas que les utilisateurs définissent leurs requêtes en termes de types de données connus. En revanche, il recherche toutes les parties de la base de données à la recherche de résultats. `msearch` est utilisé par la recherche de texte de la vue Événements. Consultez la section « Résultats du filtrage et de la recherche dans la vue Événements » dans le *Guide Investigation et Malware Analysis* pour plus d'informations sur les formulaires et exemples de recherche acceptés.

Paramètres `msearch` :

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

L'algorithme `msearch` fonctionne de la manière suivante :

1. Un ensemble de sessions est identifié à partir de l'index en trouvant les croisements entre trois ensembles :
   - (Ensemble 1) Toutes les sessions de la base de données
   - (Ensemble 2) Sessions correspondant au paramètre de la clause `where`
   - (Ensemble 3) Sessions ayant indexé des valeurs qui correspondent au paramètre de la chaîne de recherche, si la balise `si` est spécifiée.
2. Si la recherche indique le paramètre `sm` , tous les éléments méta de l'ensemble de sessions identifiées à l'étape 1 sont lus et analysés pour voir s'ils correspondent au paramètre de la chaîne de recherche. Les éléments méta sont lus à partir du service le plus proche du point où la recherche a été exécutée. Par exemple, si la recherche est réalisée sur un broker, les éléments méta peuvent être lus à partir du Concentrator le plus proche du broker, mais si la

recherche est réalisée sur un Archiver, les éléments méta seront lus à partir de l'Archiver lui-même.

3. Si la recherche spécifie le paramètre `sp` , toutes les données brutes des paquets ou entrées de log provenant de l'ensemble de sessions identifié à l'étape 1 est lu et analysé pour vérifier s'il correspond au paramètre de la chaîne de recherche. Les paquets sont lus à partir du service le plus près du point où la recherche a été exécutée. Par exemple, si la recherche est réalisée sur un Concentrator, les données de paquets sont lues à partir du Decoder, mais si la recherche est réalisée sur un Archiver, les données de paquets sont lues à partir de l'Archiver lui-même.
4. Les correspondances des étapes 2 et 3 sont renvoyées au fur et à mesure qu'elles sont trouvées, jusqu'à ce que le paramètre `limit` soit atteint. La limite spécifie le nombre maximum de sessions pour lesquelles les données de méta et de paquets sont analysées. Si aucune limite n'est spécifiée, l'intégralité de l'ensemble de sessions de l'étape 1 est analysée.

Balises `msearch`

Balise Description

`sp`	Analyse les données brutes des paquets
`sm`	Analyse toutes les données méta
`si`	Effectue des recherches d'index pour tous les paramètres de recherche avant d'analyser les méta
`ci`	Effectue une recherche non sensible à la casse. Les résultats de la recherche conservent la casse.
`regex`	Traite le paramètre de recherche en tant qu'expression régulière. Une seule expression régulière peut être spécifiée, mais l'expression régulière peut être arbitrairement complexe.

Mode de recherche d'index `msearch`

L'utilisation du mode de recherche d'index, spécifié grâce à la balise `si` , permet de renvoyer des résultats bien plus rapidement que les autres modes. La principale limite de ce mode est qu'il ne renvoie que des correspondances sur les termes du texte qui correspondent aux valeurs des méta dont la valeur est indexée.

- Le paramètre `si` doit être combiné à la balise `sm` . Le paramètre `si` implique que la recherche ne peut correspondre qu'à des méta indexées.
- Le paramètre `si` peut être utilisé avec des recherches regex, même si seules des valeurs de texte indexé sont renvoyées. Les adresses IP et les chiffres ne peuvent pas correspondre au regex.

Conseils `msearch`

- Utilisez toujours une clause `where` pour spécifier la période pour la recherche.
- Pour rechercher des plages d'adresses IP, spécifiez-les dans la clause `where` .
- Utilisez le paramètre `limit` lorsque vous n'utilisez pas le mode de recherche d'index. Si ce n'est pas le cas, la quantité de données lues par les bases de données de méta et de paquets sera extrêmement vaste.

Procédures stockées

Les appels de `query` et de `values` fournissent plus de fonctionnalités de recherche à un niveau faible. Pour les exemples d'utilisation plus avancés, il existe des procédures stockées côté serveur.

Utilisation des guillemets dans la syntaxe de requête

L'analyseur de requête ne tient pas compte du fait que des guillemets simples ou doubles soient employés dans une instruction de requête. Une valeur à guillemets simples ou doubles est traitée en tant que métadonnée de texte.

L'analyseur de requête tente de comprendre ce que vous placez dans l'instruction. Il n'est pas très strict sur les éléments acceptés.

Par exemple :

`reference.id=4752`

Cette clause identifie les sessions qui possèdent une métavaleur `reference.id` qui a une valeur *numeric* de 4752.

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

Cette clause identifie les sessions qui possèdent une métavaleur `reference.id` qui a une valeur *numeric* de `4752` .

Cependant, le moteur de requête compare implicitement les nombres et les chaînes qui ressemblent à des numéros comme étant égaux, lorsque les valeurs sont identiques sémantiquement. Il fonctionne donc avec les deux syntaxes.

Pour obtenir les meilleures performances, cependant, il est toujours conseillé de construire les requêtes de sorte que la syntaxe de requête corresponde aux types de données générés par l'analyseur.

Par exemple, si l'analyse crée `reference.id` en tant que type de données numérique (tel que `uint32` ou `uint64`), utilisez la syntaxe numérique.

Si l'analyseur crée `reference.id` en tant que type de données de texte, utilisez la syntaxe de chaîne.

Personnalisation d'index

Cette rubrique décrit comment utiliser le fichier d'index personnalisé permettant de personnaliser l'index. Chaque service NetWitness NextGen est installé avec une configuration d'index par défaut conçue pour couvrir les besoins en index de la plupart des utilisateurs du produit. Toutefois, il est possible d'indexer de nouvelles clés méta afin d'utiliser l'index avec le contenu personnalisé qui a généré les méta personnalisées.

Emplacements du fichier de configuration de l'index

Pour personnaliser l'index, il faut apporter des modifications au fichier d'index personnalisé. L'emplacement de ce fichier est **/etc/netwitness/ng/index-<nomduservice>-custom.xml** , où **<nomduservice>** correspond au nom du produit que vous personnalisez. Par exemple, le fichier d'index de Concentrator est /etc/netwitness/ng/index-concentrator-custom.xml.

Les produits Concentrator comprennent aussi un fichier décrivant la configuration de l'index par défaut : **/etc/netwitness/ng/index-concentrator.xml** . Ce fichier est utile en tant que modèle pour montrer comment le fichier d'index personnalisé est mis en forme.

Si vous personnalisez l'index dans le fichier d'index personnalisé, ces personnalisations remplacent tout conflit existant avec la configuration d'index par défaut.

Vous pouvez apporter des modifications au fichier d'index personnalisé lorsque le service fonctionne. Lorsque le service reçoit une commande save d'index, les modifications apportées au fichier d'index personnalisé sont lues et appliquées à l'index.

Les modifications apportées à l'index ne peuvent s'appliquer qu'aux nouvelles données entrantes. Les données ne peuvent pas être réindexées de manière rétroactive avec une nouvelle configuration d'index personnalisé, sauf par reconstruction de l'index .

Entrées de configuration d'index

Le fichier d'index personnalisé est un document XML. L'élément racine de ce document est l'élément `language` , puis il y a un élément par clé méta à l'intérieur du fichier pour décrire chaque index personnalisé. Chaque élément de configuration d'index personnalisé ressemble à cela :

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

Définitions de chaque attribut de cet élément : Attribut | Description -|- Nom | Nom de la clé méta qui sera indexée Description | Description compréhensible du type de méta Niveau| Type d'index qui sera créé pour cette clé méta valueMax | Valeurs uniques maximales qui seront stockées pour cette clé par tranche Format | Format des données conservé par tous les éléments méta portant ce nom de clé méta.

Les sections qui suivent passent en revue ces paramètres en détails.

Noms méta

Le nom méta utilisé par l'index fait référence au nom de clé méta présent dans chaque élément méta de la base de données méta. Ces noms méta sont générés par les Decoders lors de l'analyse. Les parsers peuvent choisir de générer des méta avec n'importe quel nom de clé méta. Ainsi, l'index personnalisé vous permet de choisir les éléments méta générés par le Decoder qui seront indexés.

Les noms de clé méta peuvent avoir une longueur de 16 caractères, et contenir uniquement des lettres ou le caractère « . ».

Type de données

Lorsque le Decoder génère des éléments méta, il leur attribue un type de données. Chaque parser peut choisir le type de données méta qu'il génère. Toutefois, il existe des types de données standard et recommandées pour chacune des clés méta par défaut. Par exemple, ip.src et ip.dst sont stockées comme le type de méta IPv4, et alias.host est stocké comme le type de méta Texte. Chaque parser doit accepter le format des données de chaque clé méta générée par le Decoder.

Lors de l'ajout d'un index personnalisé au Concentrator, le type de données de l'index personnalisé doit correspondre au format des données générées par le Decoder. Si les types ne correspondent pas, le Concentrator tente la conversion des méta générées dans le type spécifié pour l'index personnalisé. Toutefois, ces conversions ne réussissent pas toujours, et l'index qui en résulte peut produire des résultats indéfinis.

De la même façon, lorsque de nombreux Decoders et Concentrators collaborent dans une installation NetWitness, ils doivent se mettre d'accord sur les types de chaque clé méta. Des conflits de types méta entre services NetWitness NextGen peuvent engendrer un comportement indéterminé.

Le tableau suivant montre les types de métadonnées pris en charge par les services NetWitness NextGen.

Type	Taille en octets	Description
Int8	1	Nombre entier signé sur 8 bits
UInt8	1	Nombre entier non signé sur 8 bits
Int16	2	Nombre entier signé sur 16 bits
UInt16	2	Nombre entier non signé sur 16 bits
Int32	4	Nombre entier signé sur 32 bits
UInt32	4	Nombre entier non signé sur 32 bits
Int64	8	Nombre entier signé sur 64 bits

UInt64	8	Nombre entier non signé sur 64 bits
UInt128	16	Nombre entier non signé sur 128 bits
Float32	4	Nombre à virgule flottante sur 32 bits, simple précision
Float64	8	Nombre à virgule flottante sur 64 bits, double précision
TimeT	8	Horodatage epoch d'Unix
Binaire	1-255	Données binaires arbitraires
Text	1-255	Données de texte encodé UTF-8
IPv4	4	Octets d'adresse IPv4
IPv6	16	Octets d'adresse IPv6
MAC	6	Octets d'adresse MAC

Lorsque vous définissez un index personnalisé, il est important d'utiliser le meilleur type de données pour les méta. Par exemple, ne stockez jamais d'adresses IP sous forme de Texte car la représentation Texte consomme plus d'octets que la représentation IPv4.

Niveaux d'index

Il existe trois niveaux ou types d'indexation : IndexNone, IndexKeys et IndexValues.

IndexNone

Ce type d'index personnalisé n'est pas vraiment un index. Les entrées d'index personnalisé dont le niveau est IndexNone n'existent que pour définir et documenter la clé méta. Les entrées IndexNone peuvent être utilisées dans les index personnalisés Decoder afin d'appliquer un type de données spécifique pour une clé méta sur tous les parsers d'un Decoder.

IndexKeys

Ce type d'index personnalisé indique que l'index n'effectue le suivi que des sessions qui contiennent des éléments méta avec ce nom de clé méta. Toutefois, il n'indexe aucune valeur unique dans la base de données méta pour la clé méta.

Les indices de niveau clé occupent bien moins d'espace de stockage, de mémoire et de temps de gestion des CPU, mais ils requièrent beaucoup plus de travail de la part du moteur de requête lorsque vous les utilisez pour des opérations de requêtes ou de valeurs.

Si elle est utilisée dans une clause where, une clé méta indexée au niveau de la clé ne peut être utilisée que pour résoudre des opérations comme exists ou !exists.

IndexValues

Ce type d'index personnalisé conserve les sessions contenant chaque valeur unique individuelle pour la clé méta. Ce type d'index est également appelé « index complet ».

Ce type d'index est nécessaire pour le traitement efficace de la plupart des clauses where, et pour l'utilisation de cette clé méta comme paramètre fieldName d'un appel « values ».

Value Max

Value Max est un paramètre dont l'impact peut s'avérer significatif sur la précision et la performance d'un index de niveau Valeur.

Alors que le Decoder analyse les paquets ou logs, il est permis de créer des méta de n'importe quel type et de n'importe quelle valeur. Généralement, ces éléments méta sont créés à partir des données copiées directement depuis le paquet ou le log. Ainsi, n'importe qui peut créer des valeurs méta uniques en réponse à presque tous les événements.

La performance de l'index dépend directement du nombre de valeurs uniques qu'il a trouvées pour chaque clé méta. Alors que le nombre de valeurs uniques augmente, le taux auquel les nouvelles méta sont indexées peut baisser, tout comme la vitesse à laquelle les requêtes sont finalisées. Du fait que quiconque qui peut influencer la création de valeurs méta uniques, n'importe qui peut influencer la performance de l'index.

Le paramètre Value Max limite le nombre de valeurs uniques pouvant entrer dans l'index. Ainsi, un utilisateur malveillant ne peut pas inonder le système avec de nombreuses valeurs uniques dans une tentative de mettre le système NetWitness hors service.

Il est important de définir le paramètre Value Max pour toutes les clés méta dont la valeur est directement influencée par les paquets ou logs entrants.

Le paramètre Value Max s'applique uniquement aux valeurs ajoutées depuis la dernière opération « save » de l'index.

La définition de la limite supérieure du paramètre Value Max varie d'une version à l'autre et en fonction de la valeur de RAM disponible pour le service NetWitness NextGen. À partir de la version 10.3, le plafond recommandé pour Value Max est 5 000 000 pour toutes les clés méta. Si les index personnalisés sont nombreux, cette valeur devra peut-être être inférieure.

maxLength

Le paramètre max length est utilisé exclusivement sur le type de méta `word` . Il doit correspondre à la valeur de **/decoder/parsers/config/token.max.length** dans le service Log Decoder qui génère des métas de token de mot. L'index utilise le paramètre maxLength pour interpréter correctement les termes de recherche fournis à la fonction `msearch` SDK.

Changement de nom de clé

La langue de l'index prend en charge le concept de changement de nom de clé. Cette fonction est utilisée pour assurer la compatibilité en amont pour que les nouveaux noms de clés rendent obsolètes et remplacent les anciens noms de clés. Pour changer un nom, ajoutez les éléments `rename` à la clé. Cela indique la clé parent qui renomme la clé dans l'élément de changement de nom. Par exemple, la définition de clé ci-dessous définit une nouvelle clé nommée `port_src` qui renomme la clé `tcp.srcport` .

```
<key name="port_src" description="Source Port" format="UInt16"
level="IndexValues">
```

```
        <rename name="tcp.srcport"/>
</key>
```

L'élément de changement de nom indique la base de données qui utilise la clé parent, dans ce cas `port_src` inclut les éléments méta avec le type port_src, et les éléments méta avec le type tcp.srcport. Par conséquent, les nouveaux éléments méta peuvent être ajoutés à la base de données et interrogés via `port_src` , et ce type de requête renvoie des informations précédemment stockées dans tcp.srcport également.

L'élément de changement de nom accepte un seul attribut, `name` , qui fait référence à une clé précédemment définie.

Les clés indiquées par des éléments de changement de nom doivent disposer du même type que la clé parent.

Les clés indiquées par des éléments de changement de nom doivent disposer du même niveau d'index que la clé parent.

Si une clé est redéfinie dans un fichier d'index personnalisé et que la clé redéfinie contient les éléments de changement de nom, ces derniers remplacent tous les éléments de changement de nom précédemment définis.

Reconstruction de l'index.

En fonctionnement normal, les modifications apportées à la configuration d'index pour un service sont uniquement appliquées aux nouvelles données qui entrent dans la collecte. Reconstruire l'index sur toutes les données de la collecte est un processus qui prend du temps, car il nécessite la lecture de tout le stockage de la métabase de données sur le disque.

Dans les versions 11.0 et ultérieures, il est possible de reconstruire l'index lorsque le service est en ligne. Les services de la version 11.0 reconstruisent les index en arrière-plan chaque fois que le service détecte qu'une partie de la session et les métabases de données sont non indexées.

Activation de la fonction de réindexation en arrière-plan

La fonction de réindexation en arrière-plan est activé dès que le service démarre. Au démarrage, la fonction de réindexation vérifie les espaces entre les sessions indexées et les sessions présentes dans la session et la métabase de données. Si des écarts sont détectés, la fonction de réindexation en arrière-plan commence à réindexer la session et la métabase de données sur le service.

Exemples d'événements pouvant activer la fonction de réindexation en arrière-plan :

1. Une panne de courant ou un blocage s'est produit, rendant la dernière tranche de l'index corrompue. Les données corrompues sont supprimées lors du démarrage, en laissant un écart dans l'index.
2. Les données d'index sont supprimées de force, soit en réinitialisant l'index, soit en supprimant des fichiers du système de fichiers.

Contrôle de la fonction de réindexation en arrière-plan

L'activation de la fonction de réindexation en arrière-plan est contrôlée par le nœud de configuration `/index/config/reindex.enable` . En cas de paramétrage de `reindex.enable` sur true, au prochain démarrage du service, la fonction de réindexation sera activée. En cas de paramétrage de `reindex.enable` sur false, la fonction de réindexation ne sera pas activée lors du prochain démarrage du service, mais continuera à fonctionner jusqu'à ce que le service soit redémarré.

Algorithme de réindexation en arrière-plan

Le fonctionnement de la fonction de réindexation en arrière-plan s'effectue comme suit :

1. L'index examine les plages de sessions qui sont présentes dans l'index et les compare aux plages de sessions ayant des métadonnées valides. Toutes les incohérences entre les deux sont considérées comme des écarts.
2. Les écarts dans l'index sont subdivisés en tranches en fonction de la valeur actuelle de `/index/config/save.session.count`.
3. Pour chaque tranche manquante, un index temporaire est créé dans l'un des répertoires spécifiés par `/index/config/index.dir`. Les tranches sont réindexées dans l'ordre numérique inverse. Ainsi, les sessions les plus récemment collectées sont indexées en premier.
4. Une fois que la tranche est entièrement réindexée, elle est déplacée vers son emplacement valide dans l'index en ligne. Si la tranche réindexée appartient au niveau Actif, elle est déplacée vers le niveau Chaud.
5. Les données nouvellement indexées s'affichent dans le cadre de la collecte.

État de la fonction d'indexation en arrière-plan

Le nœud stat `/index/stats/updater.state` indique l'état actuel de la fonction de réindexation en arrière-plan. Ce nœud indiquera `running`, `not running` ou `failed`. Si l'état est `failed`, vérifiez le log de service pour plus d'informations de diagnostic.

Effets sur l'agrégation

Les services qui effectuent l'agrégation utilisent l'index pour effectuer le suivi des sessions ayant déjà été agrégées. Si l'index ne contient pas suffisamment d'informations pour commencer l'agrégation, l'agrégation sera mise hors ligne jusqu'à ce que suffisamment de tranches soient réindexées. Pendant ce temps, l'état d'agrégation pour le périphérique en amont indiquera qu'il attend l'agrégation.

Forcer une réindexation

Pour forcer l'index sur un service à reconstruire :

1. Assurez-vous que `/index/config/reindex.enable` a la valeur true.
2. Réinitialiser l'index à l'aide du message `reset` sur le service. Par exemple : `/concentrator/reset index=1` redémarrera le service et supprimera tous les fichiers d'index.
3. Attendez que le service redémarre. La réindexation en arrière-plan va démarrer.

4. Les données les plus récemment collectées seront disponibles pour les requêtes dès que la tranche d'index représentant ces sessions aura été réindexée.

Techniques d'optimisation

Cette rubrique décrit les techniques d'optimisation de la base de données NetWitness Core. La base de données NetWitness Core est configurée pour gérer une grande variété de charges de travail par défaut. Cependant, comme n'importe quelle technologie de base de données, ses performances peuvent être très sensibles à la nature des données d'acquisition, et à la nature des recherches que l'utilisateur effectue dans la base de données.

Seuils

Les seuils sont une optimisation utile pouvant avoir un effet négatif sur la rapidité de renvoi des résultats dans la vue Naviguer de NetWitness Suite. Les seuils sont appliqués à l'appel `values` . Pour plus d'informations sur l'appel `values` , reportez-vous à la section Requêtes .

Le seuil définit la limite de la quantité de base de données récupérée du disque afin de produire un nombre. Pour la plupart des requêtes, le nombre de sessions correspondant à la clause `where` est très grand. Par exemple, la sélection de tous les événements du log pour une heure d'exécution à 30 000 événements par seconde correspond à 108 000 000 sessions.

RSA a introduit la fonction de seuil après avoir constaté que dans la plupart des cas où un nombre de sessions est nécessaire, aucun résultat précis n'est fourni avant la toute dernière session. Par exemple, en observant les 20 premières adresses IP présentes sur la dernière heure, il n'est pas très important de savoir si le rapport indique qu'une valeur IP correspond à 10 000 000 ou 10 000 001 sessions exactement. Ici, l'estimation est plutôt juste. Dans ces scénarios, nous pouvons faire une estimation de la valeur du nombre retourné lorsque notre nombre dépasse le paramètre de seuil. Si le seuil est atteint, le nombre restant est estimé et les résultats sont triés sur la base des chiffres estimatifs, si nécessaire.

Clauses `where` complexes

La période nécessaire pour que la base de données NetWitness Core produise un résultat dépend de la complexité de la requête. Les requêtes qui correspondent directement aux index présents sur le méta peuvent être résolues rapidement, mais il est très facile d'écrire des requêtes qui ne peuvent pas être résolues rapidement. Parfois, les requêtes qui ne peuvent être renvoyées rapidement peuvent être traitées par la base de données et l'index Core différemment pour produire des résultats beaucoup plus satisfaisants pour le client.

Il est utile de connaître le *coût* relatif de chaque partie de la clause `where` . Une clause d'un coût élevé prend plus de temps à exécuter. Dans le tableau suivant, les opérateurs de requête sont classés en fonction de leur coût relatif, du plus bas au plus élevé.

Opération	Cost

exists, !exists	Constante
= , !=	Logarithme en termes de nombre de valeurs uniques pour la clé méta, linéaire en termes de nombre d'éléments uniques qui correspondent à une expression de plage
< , > , <= , >=	Logarithme en termes de recherche de valeurs uniques, mais plus susceptibles d'être linéaires puisque l'expression correspond à une large plage de valeurs
`begins` , `ends` , `contains`	Linéaire en termes de nombre de valeurs uniques pour la clé méta
`regex`	Linéaire en termes de nombre de valeurs uniques pour la clé méta avec un coût par valeur élevé

`AND` et `OR`

Lors de la construction d'une clause `where` , gardez à l'esprit que la construction de nombreux termes avec l'opérateur `AND` peut avoir un effet bénéfique sur les performances d'une requête. Plus vous pouvez utiliser de critères pour réduire le nombre de sessions correspondant à la clause `where` , plus vous réduisez le travail de la requête. En revanche, chaque clause `OR` crée un nombre plus important de sessions à traiter pour chaque requête.

En règle générale, plus les clauses `AND` sont nombreuses dans la requête, plus cette dernière s'exécute rapidement, mais plus les clauses `OR` sont nombreuses dans la requête, plus cette dernière s'exécute lentement.

Exemple d'utilisation : Correspondance avec un grand sous-réseau

Il est courant pour les utilisateurs de construire des requêtes qui tentent d'inclure ou d'exclure un sous-réseau de classe A. Ce type de requête est courant car les utilisateurs tentent d'inclure ou d'exclure certaines grandes parties de leur réseau interne de leur procédure d'enquête.

Le moteur de recherche a du mal à résoudre cette requête en utilisant uniquement les index `ip.src` ou `ip.dst` . Le problème provient du fait qu'une clause `where` comme celle-ci :

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

Doit en réalité être interprétée en :

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

Ainsi, l'index peut avoir à créer une clause `where` , avec plus de 16 millions de termes.

La solution à ce problème est d'utiliser le service Decoder pour marquer les réseaux communs pertinents à l'aide de règles d'application. Par exemple, vous pouvez créer des éléments méta avec une règle d'application qui ressemble à celle-ci :

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

Cette règle crée des éléments méta sur le réseau des clés méta avec la valeur interne pour n'importe quelle adresse IP sur le réseau 10.0.0.0/8.

La clause `where` pourrait être exprimée comme suit :

```
network = "internal"
```

En supposant qu'il y ait un index `value-level` sur le réseau méta, l'index n'a pas besoin de développer cette requête de manière complexe, et les sessions correspondant au sous-réseau souhaité sont mises en correspondance très rapidement.

Exemple d'utilisation : Correspondance de sous-chaînes

L'utilisation des opérateurs `begins` , `ends` , `contains` et `regex` dans une clause `where` peut être très lente s'il existe un grand nombre de valeurs uniques pour la clé méta. Chacun de ces opérateurs est évalué indépendamment par rapport à chaque valeur unique. Par exemple, si l'opérateur est `regex` , le `regex` doit être exécuté de façon indépendante par rapport à chaque valeur unique.

Pour contourner ce problème, la stratégie la plus efficace consiste à réorganiser les éléments méta pour que l'utilisateur n'ait pas besoin d'utiliser une correspondance de sous-chaînes.

Par exemple, si les utilisateurs tentent de trouver le nom d'hôte dans une URL, quelque part dans la session. Les utilisateurs peuvent écrire une clause `where` comme suit :

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

Dans ce scénario, il est probable que la clé méta `url` contienne une valeur unique pour chaque session qui a été capturée par le Decoder, et renferme donc un grand nombre de valeurs uniques. Dans ce cas, l'opération contains est lente.

La meilleure approche est d'identifier la partie de métadonnées que vous tentez de faire correspondre, et de déplacer la correspondance vers l'analyseur de contenu.

Par exemple, s'il y a une métadonnée en cours de génération pour chaque URL, un analyseur peut aussi décomposer l'URL en ses composants constitutifs. Par exemple, si le Decoder génère les métadonnées d'URL avec la valeur `http://www.rsa.com/netwitness` , il peut également générer des métadonnées `alias.host` avec la valeur `www.rsa.com` . Les requêtes peuvent être effectuées en utilisant :

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

Comme l'opérateur de chaîne n'est plus nécessaire, la requête est beaucoup plus rapide.

Sauvegardes d'index

L'index Core est subdivisé par points de sauvegarde, également connus sous le nom de tranches. Lorsque l'index est sauvegardé, toutes ses données sont vidées sur le disque, et cette partie de l'index est uniquement accessible en lecture seule. Les sauvegardes ont deux fonctions :

- Chaque point de sauvegarde représente un lieu où l'index pourrait être récupéré en cas de panne de courant.
- Une sauvegarde périodique peut faire en sorte que la partie de l'index qui est activement mise à jour ne soit pas plus grande que la mémoire vive (RAM).

Les points de sauvegarde ont pour effet de diviser l'index en segments indépendants qui ne se chevauchent pas. Lorsqu'une requête doit traiter plusieurs points de sauvegarde, il faut réexécuter les parties de la requête et en fusionner les résultats. Au final la requête prend plus de temps à s'exécuter.

Par défaut, pour les installations de NetWitness Suite 10.5 ou versions ultérieures, une sauvegarde est effectuée sur l'index Core chaque fois que 600 000 000 sessions sont ajoutées à la base de données. Cet intervalle est défini par le paramètre de configuration d'index `save.session.count` . Pour plus d'informations, consultez la section Limites de la configuration .

Pour les versions antérieures de NetWitness Suite, ou les systèmes ayant été mis à niveau à partir de versions NetWitness Suite antérieures à la version 10.5, utilisez un planning de sauvegarde basé sur l'heure pour sauvegarder l'index toutes les 8 heures. Vous pouvez voir l'intervalle d'enregistrement actuel en utilisant l'éditeur du planificateur dans l'interface utilisateur de NetWitness Admin pour le service. L'entrée par défaut ressemble à ce qui suit :

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

En ajustant l'intervalle, vous pouvez contrôler la fréquence de création des sauvegardes.

Effets de l'augmentation de l'intervalle de sauvegarde

En augmentant l'intervalle de sauvegarde, les points de sauvegarde sont créés moins fréquemment, ce qui réduit le nombre de points de sauvegarde. Cela a un effet positif sur les performances des requêtes, car les tranches ne sont pas traitées par les requêtes. En revanche, cela fait moins de données à analyser par les requêtes.

Il y a néanmoins des inconvénients à augmenter l'intervalle. Premièrement, le Concentrator est plus susceptible d'atteindre la limite de `valueMax` définie sur l'un des index. Deuxièmement, le temps de restauration en cas d'arrêt ou de panne de courant forcée est accru. Et troisièmement, le taux d'agrégation peut en pâtir si la tranche d'index devient trop grande pour tenir dans la mémoire.

Effets de la réduction de l'intervalle de sauvegarde

En réduisant l'intervalle de sauvegarde, il est possible d'éviter d'atteindre les limites de `valueMax` tout en conservant un index de valeur complète pour les métadonnées contenant un grand nombre de valeurs uniques. La diminution de l'intervalle de sauvegarde a un impact négatif sur les performances des requêtes, puisque plusieurs tranches sont créées.

Utiliser `valueMax`

La limitation `valueMax` peut être frustrante pour les clients qui veulent indexer tous les méta uniques possibles. Malheureusement, ce n'est pas possible de manière générale. Les clés méta peuvent avoir des données aléatoires arbitraires partout sur Internet, et toutes les valeurs uniques ne peuvent pas être indexées.

Cependant, il est possible de contourner certaines limites de `valueMax` en utilisant des index de niveau clé au lieu des index de valeur. Les index de niveau clé ne sont pas influencés par `valueMax` .

Il est possible d'utiliser la vue Naviguer sur une clé méta indexée au niveau clé. La base de données utilise des index de niveau valeur dans la clause `where` lorsque cela est possible, mais l'analyse de la base de métadonnées permet de résoudre des valeurs uniques pour l'appel `values` . Cette approche fonctionne bien lorsque la clause `where` fournit un filtre efficace pour limiter la portée de la recherche à un petit nombre de sessions, peut-être moins de 10 000 sessions.

Dans les cas où la valeur `valueMax` est atteinte, les utilisateurs peuvent effectuer une analyse de base de données sur leurs requêtes afin de s'assurer qu'aucune valeur pertinente n'a été abandonnée. Cette fonctionnalité est accessible dans le client Investigator 9.8 via le menu du clic-droit de la vue Navigation. Bien que l'analyse de la base de métadonnées prenne beaucoup de temps, elle permet au client de vérifier qu'il n'y a pas d'omissions dans son rapport.

Mise en parallèle des charges de travail

Si le client utilise de nombreux rapports, vérifiez qu'il exploite efficacement les options d'exécution parallèle disponibles dans Reporting Engine. De même, vérifiez que le nombre de `max.concurrent.queries` est approprié pour le matériel.

La vue Naviguer de NetWitness Suite est capable d'exécuter les composants de la sortie en parallèle, ce qui peut avoir un impact considérable sur les performances perçues du service NetWitness Core.

Reconstruction d'index

Dans de rares cas, un service Core pourrait bénéficier d'une reconstruction d'index. Exemples :

- Le service NetWitness Core dispose de tranches d'index créées par une très vieille version du produit et n'a pas déployé de données depuis plus de six mois.
- L'index a été configuré de manière incorrecte, et le client veut ré-indexer tous les méta avec une nouvelle configuration d'index.
- La charge du trafic sur le service Core était très légère et l'intervalle de sauvegarde était grande, ce qui générait des tranches inutiles.

Dans chacun des cas, une reconstruction d'index peut améliorer les performances. Pour cela, vous devez envoyer le message `reset` avec le paramètre `index=1` au dossier `/decoder` sur un Decoder, au dossier `/concentrator` sur un Concentrator, ou au dossier `/archiver` sur un Archiver.

Il faut savoir qu'une réindexation complète prend des jours pour s'exécuter sur un Concentrator entièrement chargé, et peut-être des semaines sur un Archiver plein.

Évolution de la rétention

Il existe plusieurs façons d'améliorer la conservation de la base de données NetWitness Core. La rétention fait référence à la période de stockage des données dans la base de données.

La première étape dans l'analyse de la rétention est de déterminer quelle partie de la base de données est le facteur limitant en termes de rétention. Les bases de données des paquets, métas et sessions fournissent les statistiques `packet.oldest.file.time` , `meta.oldest.file.time` et `session.oldest.file.time` dans le dossier `/database/stats` pour afficher l'âge du fichier plus ancien dans la base de données. L'index fournit les statistiques `/index/stats/time.begin` pour indiquer l'heure de session la plus ancienne dans l'index.

Augmentation de la rétention des paquets et des métadonnées

Le principal mécanisme pour augmenter la rétention sur ces bases de données est d'ajouter plus de stockage. S'il n'est pas possible d'ajouter du stockage au service NetWitness Core, alors il peut être nécessaire d'utiliser les options de compression sur les bases de données de paquets et les bases de métadonnées en vue de réduire la quantité de données écrites sur chaque base de données.

Si la rétention des métadonnées présente un problème, vous pouvez éventuellement supprimer le contenu inutile du Decoder générant les métadonnées. De nombreux analyseurs génèrent des méta que le client n'a pas besoin de stocker à long terme.

Augmentation de la rétention de l'index

La rétention de l'index est généralement plus longue que celle des bases de données, mais avec un index personnalisé complexe, la rétention d'index peut être plus courte. Habituellement, il est plus facile de supprimer les index de niveau de valeur inutiles dans la configuration personnalisée, mais vous pouvez également remplacer certains index de niveau de valeur par défaut par des index de niveau clé.

Il est aussi possible de faire évoluer l'index en ajoutant un stockage d'index supplémentaire. Cependant, le stockage de l'index doit être étendu en utilisant des disques SSD uniquement.

Évolution à l'horizontale

À partir de la version 10.3, les Concentrators et les Archivers ont la capacité de se regrouper à l'aide de l'agrégation de groupes. L'agrégation de groupes permet à un seul Decoder de fournir des sessions à plusieurs Concentrators ou Archivers avec une charge équilibrée. L'agrégation de groupes permet à la charge applicative de la requête et de l'agrégation d'être répartie sur un large pool de matériel de manière arbitraire.

Pour plus d'informations, consultez la section « Agrégation de groupes » dans le *Guide de déploiement* .

Regroupement des charges applicatives

La base de données NetWitness Core fonctionne beaucoup mieux lorsque tous les utilisateurs du système travaillent dans la même zone de la base de données. Étant donné que la base de données est alimentée en données par le Decoder sur la base du premier entré, premier sorti, les données dans la base de données sont généralement regroupées en fonction de l'heure de capture et de stockage. Par conséquent, la base de données fonctionne mieux lorsque tous les utilisateurs travaillent sur la même période de données.

Il n'est pas toujours possible pour tous les utilisateurs de travailler sur la même période simultanément. La base de données NetWitness Core peut gérer ce cas d'utilisation, mais l'opération est lente, car elle doit alterner entre plusieurs périodes différentes dans la mémoire vive (RAM). Il est impossible d'avoir toutes les périodes de temps dans la RAM en même temps. Généralement, moins de 1 % de la base de données et moins de 10 % de l'index occupent la RAM.

Pour que NetWitness Suite fonctionne pour le client, il est important que le client organise ses utilisateurs en groupes qui ont tendance à travailler sur les mêmes plages horaires. Par exemple, les utilisateurs qui font un suivi quotidien des données les plus récentes peuvent constituer un groupe d'utilisateurs. Un autre groupe peut se former avec les utilisateurs qui émettent des requêtes par la suite dans le cadre d'une procédure d'enquête. Ou encore un autre groupe d'utilisateurs créent des rapports sur de grandes périodes de temps. Tenter de servir tous les groupes à partir d'une seule base de données peut générer une frustration et une longue attente avant que les résultats ne soient produits. Toutefois, si les différents cas d'utilisation peuvent être répartis sur plusieurs services Concentrator, les performances seront beaucoup mieux perçues. Dans ce cas, il peut être bénéfique d'utiliser plusieurs services Concentrator avec moins de RAM et de CPU plutôt qu'un seul Concentrator volumineux et coûteux destiné à répondre à tous les besoins.

Période de cache

Prenons par exemple cette séquence d'événements :

1. À 9h00, l'utilisateur « kevin » se connecte à un Concentrator et demande un rapport sur la dernière heure de collecte.
2. Le Concentrator récupère les rapports pour la période comprise entre 08:00 et 09:00.
3. À 09h02, l'utilisateur « scott » se connecte au même Concentrator et demande également un rapport sur la dernière heure de collecte.
4. Le Concentrator récupère les rapports pour la période comprise entre 08:02 et 09:02.

Notez que même si les deux utilisateurs consultent les rapports à des plages de temps rapprochées, le travail effectué par le Concentrator pour produire les rapports pour Kevin ne permet pas de renvoyer les rapports à Scott, puisque les plages horaires sont légèrement différentes. Le Concentrator doit recalculer la plupart des rapports pour Scott.

Le paramètre `cache.window.minutes` sur le nœud `/sdk` vous permet d'optimiser cette situation. Lorsqu'un utilisateur se connecte, le point dans le temps représentant les données les plus récentes de la collecte se déplace uniquement vers l'avant par incréments du nombre de minutes définies par ce paramètre.

Par exemple, supposons que `/sdk/config/cache.window.minutes` ait la valeur 10\. Réévaluer l'action ci-dessus modifie la séquence des événements.

1. À 9h00, l'utilisateur « kevin » se connecte à un Concentrator et demande un rapport sur la dernière heure de collecte.
2. Le Concentrator récupère les rapports pour la période comprise entre 08:00 et 09:00.
3. À 09h02, l'utilisateur « scott » se connecte au même Concentrator et demande également un rapport sur la dernière heure de collecte.

4. Le Concentrator récupère les rapports pour la période comprise entre 08:00 et 09:00.
5. À 09h10, l'utilisateur « scott » recharge les rapports sur la dernière heure de collecte.
6. Le Concentrator récupère les rapports pour la période comprise entre 08:10 et 09:10.

Le rapport retourné à l'étape 3 tombe dans la période de cache, il est donc retourné instantanément. Cela donne à Scott l'impression que le Concentrator est très rapide.

Ainsi, de plus grands paramètres `cache.window` améliorent les performances perçues, au risque de générer de légers retards, jusqu'à ce que les dernières données soient disponibles pour la recherche.

Limites de temps

Lorsqu'une requête est en cours d'exécution sur la base de données NetWitness Core pour une longue durée, le service Core consacre de plus en plus de temps CPU et de RAM à cette requête pour qu'elle se termine plus rapidement. Cela peut avoir un impact négatif sur d'autres requêtes et sur l'agrégation. Pour éviter que les utilisateurs avec des privilèges inférieurs utilisent plus de ressources que celles allouées par leur part du service Core, il est judicieux de limiter la durée d'exécution des requêtes par les utilisateurs normaux.

Annexe A : Statistiques

Cette rubrique décrit les statistiques utilisées pour surveiller le fonctionnement du système. Les services Core fournissent un très grand nombre de statistiques pour surveiller le fonctionnement du système. Certains d'entre eux sont utiles pour le suivi des performances, alors que d'autres servent à surveiller le fonctionnement du système ou à des fins de débogage.

Statistiques présentes sous /database/stats

Le tableau suivant indique la signification des statistiques présentes sous **/database/stats** .

Statistiques	Signification
`meta.bytes`, `packet.bytes`, `session.bytes`	Taille totale des données (en octets) stockées dans chaque base de données
`meta.first.id`, `packet.first.id`, `session.first.id`	Premiers ID de métadonnées, ID de paquet et ID de session, respectivement, stockés dans la base de données
`meta.last.id`, `packet.last.id`, `session.last.id`	Derniers ID de métadonnées, ID de paquet et ID de session, respectivement, stockés dans la base de données
`meta.oldest.file.time`, `packet.oldest.file.time`, `session.oldest.file.time`	Date de création du fichier le plus ancien dans chaque base de données
`meta.rate`, `packet.rate`, `session.rate`	Nombre d'objets de métadonnées, de paquets et de sessions ajoutés à chaque base de données au cours de la dernière seconde
`meta.total`, `packet.total`, `session.total`	Nombre total d'objets de métadonnées, de paquets et de sessions dans chaque base de données
`meta.volume.bytes`, `packet.volume.bytes`, `session.volume.bytes`	Taille approximative du volume total (en octets) de tous les répertoires utilisés par chaque base de données
`meta.free.space`, `packet.free.space`,	Espace total inutilisé approximatif (en octets) de tous les répertoires utilisés par chaque base de

session.free.space	données

Statistiques présentes sous /index/stats

Le tableau suivant indique la signification des statistiques présentes sous **/index/stats** .

Statistiques	Signification
checkpoint.page , checkpoint.summary	Derniers objets stockés lors de la dernière sauvegarde d'un index (débogage)
index.bytes	Mesure approximative de la quantité d'espace disque requise par les fichiers d'index
index.last.load.time	Horodatage du chargement de la configuration d'index actuelle issue des fichiers de configuration d'index
memory.used	Mesure approximative de la quantité de mémoire occupée par l'index
page.first.id , summary.first.id	Objet de la première page et du résumé stockés dans l'index (débogage)
page.last.id , summary.last.id	Objet de la dernière page et du résumé stockés dans l'index (débogage)
page.total , summary.total	Nombre de pages et de résumés contenus dans l'index (débogage)
session.first.id	ID de la première session indexée
session.last.id	ID de la dernière session indexée
sessions.since.save	Nombre de sessions actuellement gérées par la tranche d'index actuelle
values.added	Nombre de valeurs uniques ajoutées à la tranche d'index actuelle
slices.total	Nombre de tranches contenues dans l'index
time.begin	Le premier méta time indexé
time.end	Le dernier méta time indexé
updater.state	L'état de la fonction de réindexation en arrière-plan

Statistiques présentes sous /sdk/stats

Le tableau suivant indique la signification des statistiques présentes sous **/sdk/stats**

Statistiques	Signification
`cache.window.time.begin`	Début de l'heure en cours appliquée par cache.window.minutes
`cache.window.time.end`	Fin de l'heure en cours appliquée par cache.window.minutes
`queries.active`	Nombre de requêtes en cours d'exécution dans l'index
`queries.queued`	Nombre de requêtes en attente d'exécution
`values.calls`	Nombre d'appels effectués dans la fonction « values » depuis le démarrage du processus
`values.calls.cached`	Nombre d'appels effectués dans la fonction « values » ayant été résolus par le cache des résultats d'appels de valeurs

Statistiques par requête

Les opérations SDK, telles que les requêtes et valeurs, fournissent des informations sur leur état d'exécution dans `/sdk/config/stats/queries/<handleid>` , où `<handleid>` représente un identifiant unique pour l'opération de requête.

Le tableau suivant indique la signification des statistiques par requête.

Statistiques	Signification
channel.path	Ces statistiques fournissent un lien au canal de connexion via lequel l'opération communique. Ce canal permet de renvoyer les résultats au client.
`query.type`	Type d'opération en cours d'exécution, telles que les requêtes ou les valeurs
`query`	Ensemble complet de paramètres attribués à la requête
`query.progress`	Pourcentage d'exécution en cours de la requête
`query.status`	Message décrivant le niveau d'exécution en cours de la requête
`running.since`	Heure de début d'exécution de la requête
`user`	Nom de l'utilisateur qui a exécuté la requête

Annexe B : Fonction inspect de l'index

L'index de base de données NetWitness Core inclut la fonction de débogage intégrée `inspect` qui fournit des informations détaillées sur la composition de ses index. La fonction inspect de l'index se trouve dans `/index/inspect` dans l'arborescence de chaque service Core. Les services dépourvus d'index comme Broker ne sont pas dotés de la fonction `/index/inspect` .

Paramètres

Options

Type : Chaîne Ce paramètre peut être défini sur la valeur `all-slices` pour collecter des informations inspect sur toutes les tranches de l'index. S'il n'est pas défini, les informations relatives à la tranche la plus récente sont renvoyées.

La collecte d'informations sur l'ensemble des tranches peut être longue si les tranches d'index sont nombreuses.

Réponse

La fonction inspect renvoie un grand nombre de lignes de paires de valeurs clés correspondant à l'état de l'index.

Synthèse des tranches

La première ligne renvoyée pour chacune des tranches est une synthèse reprenant les valeurs suivantes.

`session1`	Premier ID de session indexé dans la tranche
`session2`	Dernier ID de session indexé dans la tranche
`meta1`	Premier métaID de la première session indexé dans la tranche
`meta2`	Dernier métaID de la dernière session indexé dans la tranche

Synthèse par index

Des lignes de synthèse par index sont renvoyées pour chacun des index. Seuls les index de niveau Valeur sont consignés.

`key`	Nom de la métaclé de l'index
`pathname`	Chemin de l'index sur le disque

`values`	Nombre de valeurs uniques stockées dans l'index
`summaries`	Nombre d'entrées de synthèse occupées par l'index dans le fichier summary.db
`pages`	Nombre d'entrées de page occupées par l'index dans le fichier page.db
`sessions`	Nombre de sessions ayant comporté une valeur qui a été insérée dans l'index
`size`	Cumul des métavaleurs de taille de l'ensemble des sessions ayant inséré une valeur dans l'index
`packets`	Nombre cumulé des paquets de l'ensemble des sessions ayant inséré une valeur dans l'index
`summary1`	Premier ID de synthèse utilisé par l'index
`summary2`	Dernier ID de synthèse utilisé par l'index
`session1`	Premier ID de session référencé par l'index
`session2`	Dernier ID de session référencé par l'index

Pied de page de la synthèse des tranches

La dernière ligne de chaque rapport inspect contient les statistiques cumulatives de tous les index compris dans la tranche.

`totalKeys`	Nombre de métatypes indexés
`totalValues`	Nombre de valeurs uniques faisant l'objet d'un suivi par tous les index compris dans la tranche
`totalMemory`	Total approximatif de la mémoire nécessaire pour ouvrir cette tranche d'index