**Graylog**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DESC** | | | | |
| Gestionnaire/Monitoring logs | | | | |
| **FILES** | | | | |
| /etc/ | graylog/ | server/ | server.conf | Configuration |
|  |  |  | Log4j2.xml | Logging configuration |
|  | Default/ | Graylog-server | JVM settings | |
| /var/ | log/ | graylog-server/ | server.log | Logs |
|  | Lib/ | Graylog-server/ | journal | Message journal files |
| **QUICK START** | | | | |
| * Configuration server.conf * Configuration elasticsearch.yml   + Cluster.name : graylog   + Network.host :   + http.port : 9200 * Configuration Graylog webserver basique :   + Configurer un Input   + Utiliser un extractor pour parser le message entrant * Configuration Graylog webserver avancée :   + Crée des streams avec indices différents pour chaque log différents   + Crée des processing pipelines pour changer les messages suivant des conditions   + Penser à configurer Message filter chain avant processing pipelines si les processing pipelines doivent s’appliquer a des champs extraits par les extracteurs | | | | |

**Schéma fonctionnel**

Pipeline rules

Stream rules

ES

Processing pipeline

Extracteur

Stream 3

Stream 2

Stream 1

Graylog

Filebeat

**Schéma fonctionnel d’affichage**

Dashboard

Widgets

Result

Lookup table

Pipeline decorator

Syslog mapper

Format string



Search

**Généralités :**

* Graylog utilise :
  + MongDB pour stocker la configuration
  + Elasticsearch pour stocker les données
  + Graylog-server avec serveur web intégré si besoin
  + Collector-sidecar si besoin de rapatrier des fichiers vers Graylog
* Graylog Marketplace pour les add-ons.
* Mot de passe root uniquement définit dans la configuration
* Inspection de la base ES par l’onglet search, puis transformation en widgets directement dans l’onglet Dashboard
* Web Interface et collector-sidecar communiquent avec Graylog-server via la REST API
* Sécurité :
  + Utiliser TLS
  + MongoDB et ES ne doivent pas écouter sur une interface publique
  + SELinux
* **Inputs :** 
  + Syslog
  + GELF
  + AWS
  + Beats via collector-sidecar
  + JSON
  + Netflow
  + Plain/Raw text
* **Extracteurs :** Parsing du message entrant
* **Stream :** routage des paquets entrants dans des catégories en temps réel et re-routés ou affectés à un set d’indices

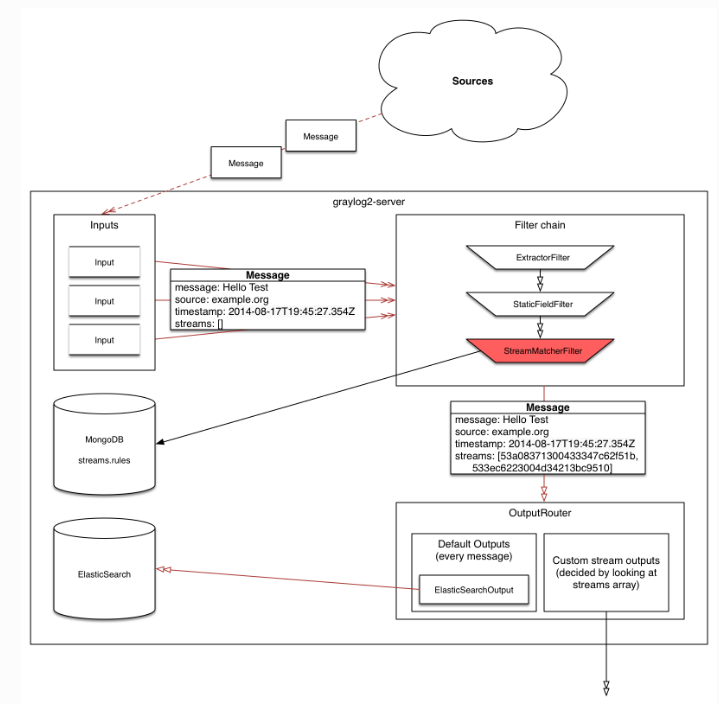
Le routing vers des streams s’effectue via des **règles** sur les messages entrants

On peut créer plusieurs stream en fonction de **conditions** (ex 1 stream d’alerting, un stream syslog, un stream d’alertes Suricata, un stream d’alertes Suricata de niveau 1…)

Chaque stream est affecté à un set d’indices.

Use case :

* Créer une liste de logins SSH failed et utilisation de quick values pour voir quel username ont été failed
* Autre



* **Outputs :** Pour forwarder des streams ailleurs
* **Alerting :** Basé sur les streams
  + Condition (Basé sur le nombre d’occurrences, le temps, le dépassement de seuils…)
  + Notification
* **Décorateurs :** Messages non modifiés en base mais modifiés à l’affichage
  + Modification de champs pour une meilleure visibilité
  + Combinaison de champs
  + Ajouts de champs aux messages

Les décorateurs sont composés de :

* Syslog severity mapper : Valeur numérique du level changé en phrase.
* Format string : Ajout d’un nouveau champ, qui peut contenir un ou plusieurs champs déjà existants, ainsi que du texte.
* Pipeline decorator
* Lookup table
* **Pipeline :** Peut être appliqué à un ou plusieurs stream

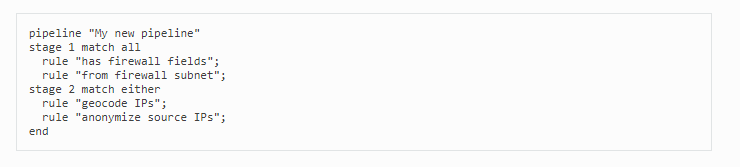
Etablissement de règles ou série de règles sur des logs : routing, blacklisting, enrichissement ou modification de logs.

2 sortes de pipeline :

* Pipeline decorator, non persistants (qui ont besoin d’un processing pipeline non affecté à un stream)
* Processing pipeline affecté à un stream, persistant

Pipeline constitué de :

* + Stages exécutés les uns à la suite des autres et possédant des priorités. **Si un stage fail, les autres étages du pipeline ne sont pas exécutés.**
    - Chaque stage possède des règles Drools : Conditions suivis d’une liste d’actions





$message.dst\_ip -> accès aux champs du message

Cf. rules plus bas

**Les règles de stream sont hard-coded donc très rapide, tandis que les règles des pipelines sont interprétées.**

* **Configuration :**

L’ordre des unités est important :

* + Message filter chain :
    - Mise en place de champs statique
    - Utilisation des extracteurs
    - Utilisation des règles de stream
  + Pipeline processor : les règles des pipelines

Ou bien on met le stream general sans règle et les processing pipelines s’occupent de les router dans les différents stream, ou bien plusieurs stream et les processing pipelines affectés à chacun des stream

* **Lookup tables :**

Transposition d’un champ en un autre via une table

Exemple : une table ip -> Host, la lookup table convertira les ips en noms.

On peut ajouter ou modifier des champs.

La table peut être :

* + Un fichier CSV

4 composants :

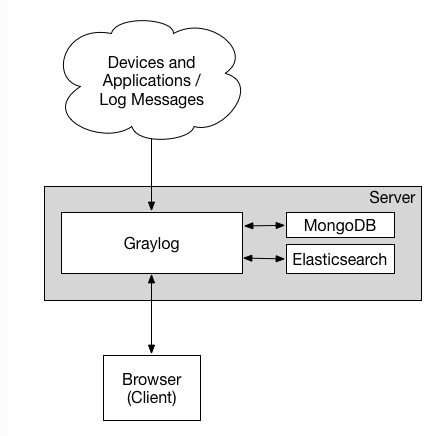
* + Data adapters s’occupe de la phase de recherche des correspondances
    - Ils lisent dans un fichier CSV
    - Peuvent se connecter à une BDD
    - Exécuter des requêtes http pour recevoir les résultats
    - Requête DNS ?
  + Caches : caching des résultats des data adapters afin d’éviter de surcharger la BDD locale.
  + Lookup tables : Lie un data adapter à un cache
  + Lookup results : Résultat retourné par une lookup table et peut être :
    - Une valeur unique exemple : correspondance IP -> host
    - Plusieurs valeurs exemple : ip -> longitude, latitude

Utilisation possible des lookup tables dans :

* + Les Extracteurs
  + Les décorateurs
  + Les pipelines via la fonction lookup() et lookup\_value()

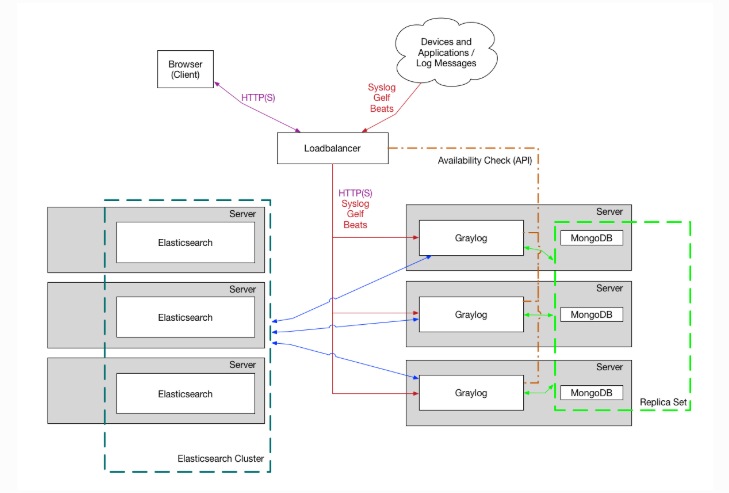
**Fonctionnement**

* Basique :

****

* Multi-node simple :
  + Elasticsearch, MongoDB et graylog-server sur leur host respectif
  + Elasticsearch d’un côté, MongoDB et graylog-server de l’autre
* Multi-node complexe :

High availability, Elasticsearch et MongoDB sont clustérisés.

****

**Fichier de configuration :**

|  |  |  |
| --- | --- | --- |
| **Directive** | **Comments** | |
| **GENERALE** | | |
| **Is\_master** | Un seul maitre Graylog-server si cluster  Ce maitre va exécuter des actions de maintenances périodiques | |
| **Node\_id\_file= <path>** | Fichier contenant l’ID de ce nœud Graylog | |
| **Root\_username** | Nom du root | |
| **Dashboard\_widget\_default\_cache\_time** | Délai de rafraîchissement des widgets par défaut | |
| **Allow\_highlighting** | Autorise le highligh des résultats lors d’une recherche Graylog, mais consomme beaucoup de ressources | |
| **STREAM** | | |
| **Stream\_processing\_timeout** | Renvoie une faute si un la durée de match d’un stream excède cette valeur | |
| **Stream\_processing\_max\_faults** | Désactivation du stream si le nombre de fautes dépasse cette valeur | |
| **DROOLS RULES (PIPELINES)** | | |
| **Rules\_files = <path>** | Drools rule file (rewriting des messages entrants)  Fichiers .drl | |
| **OBLIGATOIRE** | | |
| **Password\_secret=<secret>** | Utilisé pour le chiffrement et le salage  Min 64 caractères  Utilisé le même pour tous les nœuds  pwgen –N 1 –s 96 | |
| **Root\_password\_sha2** | Mot de passe root  Echo –n <password> | shasum –a 256  Echo –n <password> | sha256sum | |
| **Rest\_listen\_uri** | Doit être atteignable par les autres nœuds Graylog-server ainsi que par les collecteurs s’ils sont utilisés  @IP interne de Graylog  127.0.0.1 :9000/api/  **Mais mettre en publique si besoin du web server ?** | |
| **Rest\_transport\_uri** | @IP publique de Graylog  192.168.1.1 :9000/api/ | |
| **Web\_enable** |  | |
| **Web\_listen\_uri** | @IP interne de Graylog  127.0.0.1 :9000/  **Mais mettre en publique si besoin du web server ?** | |
| **Web\_endpoint\_uri** |  | |
| **Elasticsearch\_hosts** | Liste de nœuds ES ou Graylog va se connecter | |
| **Mongodb\_uri** | Connection à MongoDB | |
| **Root\_timezone= <TZ>** |  | |
| **SECURITE** | | |
| **REST** | **Rest\_enable\_tls** | HTTPS pour la REST API |
| **Rest\_tls\_cert\_file** | Certificat X509 |
| **Rest\_tls\_key\_file** | Clef privée |
| **Rest\_tls\_key\_password** | Passphrase protégeant la clef privée |
| **Web** | **web\_enable\_tls** |  |
| **web\_tls\_cert\_file** |  |
| **web\_tls\_key\_file** |  |
| **web\_tls\_key\_password** |  |
| **Email** | Voir partie correspondante | |

**Syntaxe de la recherche :**

|  |  |
| --- | --- |
| **Directive** | **Comments** |
| (ssh login) | Inclut le terme ssh ou login |
| "ssh login" | Inclut le term exact ssh login |
| \_exists : |  |
| /REGEX/ |  |
| NOT AND OR |  |
| ? \* | Globbing (doit être activé dans la configuration) |
| > | < | <= | >= |  |

**Rules :**

Rule "<name>"

when

<condition> [&& <condition2>] # AND | && | OR | || | NOT | ! | < > <= >= == !=|true

<action>

End

#si name est une variable, name.field est son champs

|  |  |
| --- | --- |
| **Fonctions générales** | **Comments** |
| To\_string() |  |
| To\_bool() |  |
| To\_double() |  |
| To\_long() |  |
| to\_ip(string) |  |
| To\_url() |  |
| Parse\_date(string, "Template") | Renvoie un objet date avec tous les champs |

|  |  |
| --- | --- |
| **Condition** | **Comments** |
| Has\_field("dst\_ip") | Le message possède le champs |
| Cidr\_match("1.1.1.1/1", <dst\_ip> |  |
| Is\_null() |  |
| Is\_not\_null() |  |
| Contains(string, mot) | Check si string contient mot |

|  |  |
| --- | --- |
| **Action** | **Comments** |
| Let name = value ; | Définition d’une variable |
| Set\_field("nom", valeur) |  |
| Rename\_field(old,new) |  |
| Remove\_field(champs) |  |
| Drop\_message() | Message droppé |
| Route\_to\_stream(id) | Les pipelines du nouveau stream seront activés sur ce message |