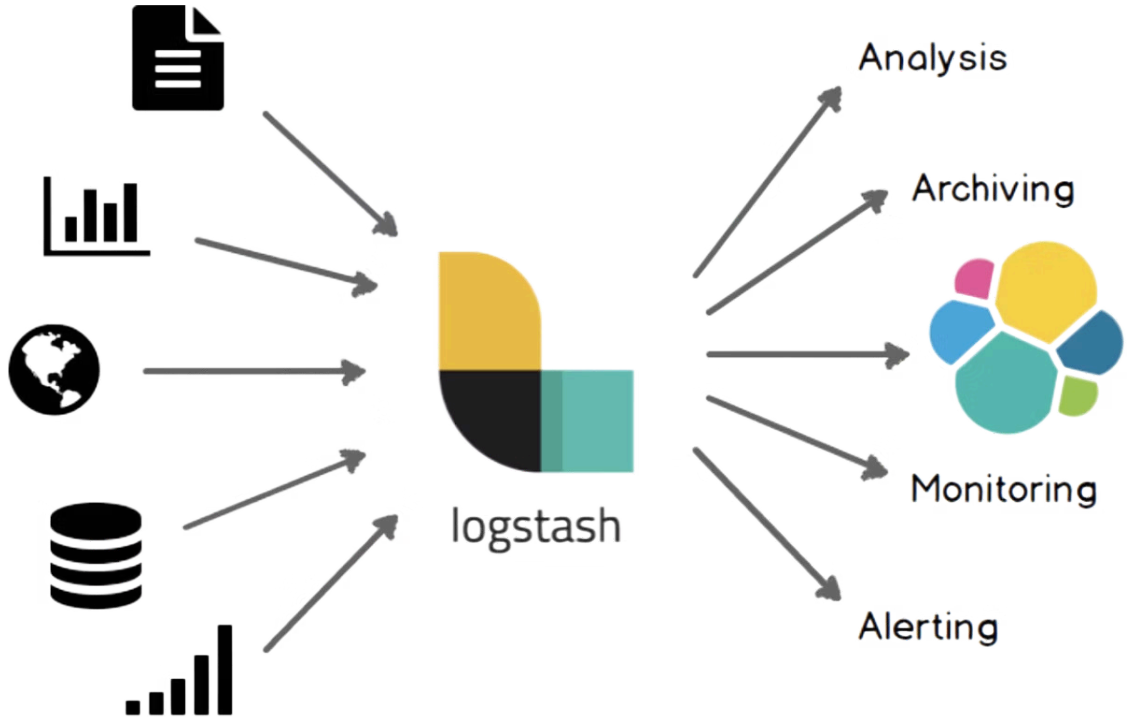
Logstash avec Elasticsearch



Rôle de Logstash dans l'Elastic Stack

Logstash est un moteur de traitement de données en temps réel qui collecte, transforme et achemine les événements vers diverses destinations. Il agit comme un pont entre les sources de données et Elasticsearch.

Dans l'Elastic Stack, Logstash se positionne entre les collecteurs (Filebeat, Metricbeat) et Elasticsearch. Il enrichit, normalise et structure les données brutes avant leur indexation.

Fonctions principales

- Ingestion multi-sources (logs, métriques, événements)
- Transformation et enrichissement des données
- Routage intelligent vers multiples destinations
- Gestion de la résilience et du buffering

Pipeline Logstash : flux de traitement

Input

Réception des données depuis diverses sources

Filter

Transformation, enrichissement et structuration

Output

Envoi vers Elasticsearch ou autres destinations

Le pipeline fonctionne en mode streaming avec gestion de buffer. Chaque événement traverse séquentiellement les trois phases. La configuration détermine le comportement à chaque étape.

```
input {
  beats { port => 5044 }
}
filter {
  grok { ... }
}
output {
  elasticsearch { ... }
}
```

Cette structure modulaire permet une grande flexibilité : plusieurs inputs peuvent converger vers un même filter, et un output peut distribuer vers plusieurs destinations.

Nouveautés Elasticsearch 9

1

ES|QL

Nouveau langage de requête unifié pour interroger les données avec une syntaxe simplifiée inspirée de SQL. Remplace progressivement les queries DSL complexes.

TSDB amélioré

Optimisations pour les séries temporelles avec compression accrue et performances de requête multipliées par 3 sur les données métriques.

Sécurité par défaut

HTTPS et authentification activés automatiquement lors de l'installation. Certificats auto-signés générés lors du premier démarrage.

Intégration Logstash → Elasticsearch

L'intégration entre Logstash et Elasticsearch 9 requiert une configuration sécurisée obligatoire. Trois éléments sont indispensables : authentification utilisateur, connexion SSL/TLS, et stratégie d'indexation.

Authentification

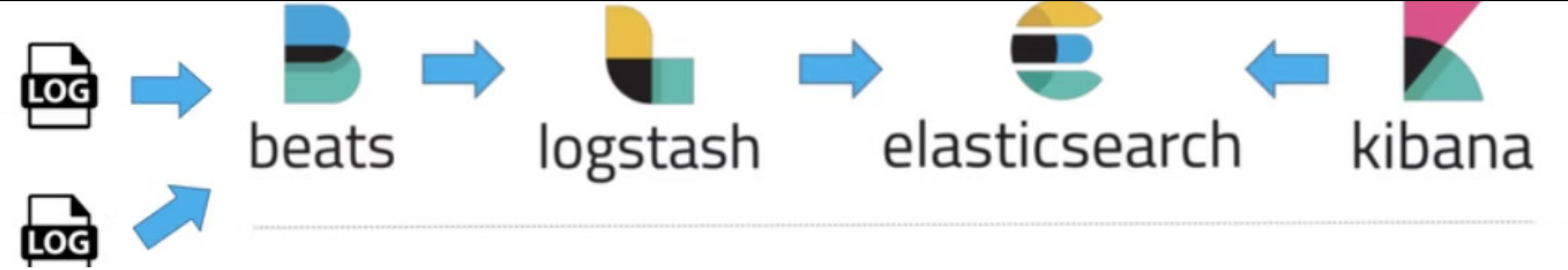
Utilisateur dédié avec rôle limité aux opérations d'écriture sur les index ciblés. Stockage du mot de passe via variable d'environnement.

SSL/TLS

Certificat CA requis pour valider l'identité du cluster Elasticsearch. Connexion chiffrée sur le port 9200.

Indexation

Définition du pattern d'index avec rotation temporelle. Support de l'ILM pour gestion du cycle de vie.



Architecture complète du pipeline

L'architecture type d'un déploiement Logstash s'articule autour de plusieurs composants interconnectés. Filebeat collecte les logs sur les serveurs applicatifs et les transmet à Logstash via le protocole Beats sur le port 5044.

Logstash traite les événements et les indexe dans Elasticsearch. Kibana interroge Elasticsearch pour la visualisation. APM Server peut également alimenter le même cluster pour les traces applicatives.

Cette architecture permet une séparation des responsabilités : collecte légère sur les serveurs, traitement centralisé dans Logstash, stockage et recherche dans Elasticsearch.

Composants de l'architecture

Filebeat

Agent léger installé sur chaque serveur source. Lit les fichiers de logs et envoie les lignes à Logstash avec back-pressure automatique.

Logstash

Serveur centralisé de traitement. Parse, enrichit et transforme les événements avant indexation. Gère le buffering pour absorber les pics.

Elasticsearch

Cluster de stockage et recherche. Indexe les documents structurés et offre des capacités de requête temps réel avec agrégations.

Kibana

Interface de visualisation et d'administration. Permet la création de dashboards, la recherche ad-hoc et la configuration du monitoring.

Gestion du buffer et des workers

Logstash utilise un système de queue interne pour gérer le flux d'événements entre les phases du pipeline. Deux types de queue sont disponibles : memory (par défaut, rapide mais volatile) et persisted (sur disque, résilient).

Le paramètre `pipeline.workers` définit le nombre de threads parallèles pour les filtres. Valeur recommandée : nombre de CPU cores. Le `pipeline.batch.size` contrôle le nombre d'événements traités par lot (défaut : 125).

```
queue.type: persisted
queue.max_bytes: 1gb
pipeline.workers: 4
pipeline.batch.size: 125
```

La queue persistante écrit sur disque avant traitement, garantissant la non-perte de données en cas de redémarrage. Compromis : latence légèrement supérieure mais résilience accrue.

Paramètres clés

- **queue.type** : memory ou persisted
- **queue.max_bytes** : taille maximale du buffer
- **pipeline.workers** : parallélisme des filtres
- **pipeline.batch.size** : événements par lot
- **pipeline.batch.delay** : attente avant envoi du lot

Configuration principale : logstash.yml

Le fichier `/etc/logstash/logstash.yml` contient la configuration globale de l'instance Logstash. Il définit les paramètres système, les chemins d'accès et les réglages de performance.

```
path.data: /var/lib/logstash
path.logs: /var/log/logstash
pipeline.workers: 4
pipeline.batch.size: 125
queue.type: persisted
queue.max_bytes: 1gb
http.host: "0.0.0.0"
http.port: 9600
```

path.data : répertoire de stockage des données persistantes (queue, plugins). **pipeline.workers** : nombre de threads pour traitement parallèle. **queue.type: persisted** : active la queue sur disque pour résilience. **http.port: 9600** : API de monitoring et statistiques.

En production, privilégier queue persistante sur volume séparé avec I/O rapides. Ajuster workers selon charge CPU observée.

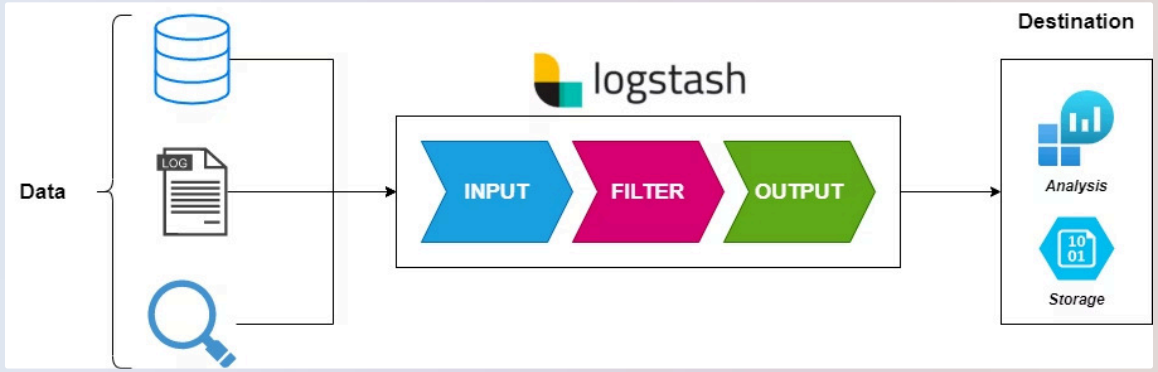
Configuration du pipeline

Les pipelines Logstash se définissent dans des fichiers de configuration situés dans `/etc/logstash/conf.d/`. Chaque fichier contient les sections input, filter et output.

```
# Ce pipeline reçoit des logs via Beats, parse les lignes Apache avec Grok,
# normalise le timestamp,puis indexe dans Elastic avec rotation quotidienne.
input {
  beats {
    port => 5044
    ssl => true
    ssl_certificate => "/etc/logstash/certs/server.crt"
    ssl_key => "/etc/logstash/certs/server.key"
  }
}

filter {
  grok {
    match => { "message" => "%{COMBINEDAPACHELOG}" }
  }
  date {
    match => [ "timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ]
    target => "@timestamp"
  }
}

output {
  elasticsearch {
    hosts => ["https://es01:9200"]
    user => "logstash_writer"
    password => "${LOGSTASH_ES_PASSWORD}"
    ssl => true
    cacert => "/usr/share/logstash/config/certs/ca.crt"
    index => "logs-%{+YYYY.MM.dd}"
  }
}
```



Inputs : sources de données

Beats

Protocole natif Elastic pour recevoir données depuis Filebeat, Metricbeat, Packetbeat. Port par défaut : 5044. Support SSL natif.

```
input {
  beats { port => 5044 }
}
```

TCP/UDP

Écoute sur socket réseau pour logs bruts. Format JSON recommandé. Utile pour applications legacy ou syslog.

```
input {
  tcp {
    port => 5000
    codec => json
  }
}
```

HTTP Poller

Interrogation périodique d'API REST. Récupère données JSON selon schedule défini. Authentification supportée.

```
input {
  http_poller {
    urls => {
      api => "https://api.exemple.fr/metrics"
    }
    schedule => { cron => "* * * * *" }
  }
}
```

Kafka

Consommation de topics Kafka. Permet architecture distribuée et découplage. Gestion automatique des offsets.

```
input {
 kafka {
 bootstrap_servers => "kafka:9092"
 topics => ["logs"]
 group_id => "logstash"
 }
}
```

Inputs pour test et développement

stdin

Entrée standard pour tests interactifs. Permet de saisir manuellement des événements et observer le traitement en temps réel.

```
input {
  stdin {
    codec => json
  }
}
```

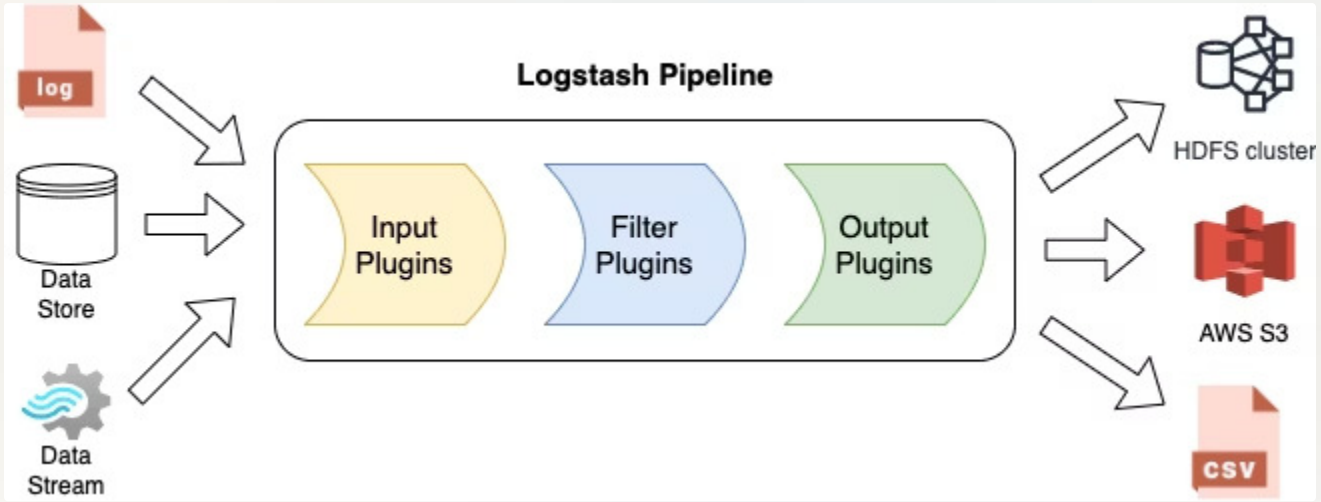
Usage : lancer Logstash avec ce pipeline, saisir JSON ligne par ligne. Idéal pour valider filtres rapidement.

file

Lecture de fichiers logs locaux. Suit la croissance du fichier (tail -f). Mémorise position pour reprendre après redémarrage.

```
input {
  file {
    path => "/var/log/app/*.log"
    start_position => "beginning"
    sincedb_path => "/dev/null"
  }
}
```

En production, préférer Filebeat pour collecte distribuée. file input surtout pour tests locaux ou petits volumes.



Filtres : transformation des données

Les filtres constituent le cœur du traitement Logstash. Ils structurent, enrichissent et normalisent les événements bruts. L'ordre des filtres est déterminant : ils s'appliquent séquentiellement.

Chaque filtre modifie l'événement en place. Un événement est un document JSON avec des champs. Les filtres ajoutent, suppriment ou transforment ces champs.

Grok : extraction de données structurées

Grok permet de parser des logs non structurés en utilisant des patterns regex nommés. Il transforme une ligne de texte brut en champs exploitables.

```
filter {
  grok {
    match => {
      "message" => "%{COMBINEDAPACHELOG}"
    }
  }
}
```

Pattern COMBINEDAPACHELOG extrait automatiquement : IP client, timestamp, méthode HTTP, URL, code status, user-agent.

Patterns disponibles dans `/usr/share/logstash/vendor/bundle/jruby/*/gems/logstash-patterns-core-*/patterns/`. Possibilité de créer patterns custom.

```
# Pattern custom
filter {
  grok {
    match => {
      "message" => "%{IP:client} \[%{HTTPDATE:timestamp}\] %{WORD:method}"
    }
  }
}
```

Patterns courants

- `%{IP:client_ip}`
- `%{NUMBER:duration}`
- `%{WORD:log_level}`
- `%{TIMESTAMP_ISO8601:ts}`
- `%{GREEDYDATA:message}`

Bonnes pratiques

Tester patterns avec Kibana Dev Tools Grok Debugger. Éviter %{GREEDYDATA} en début de pattern (performance). Nommer tous les champs extraits.

Date : normalisation des timestamps

Le filtre date parse les timestamps et les convertit en format ISO8601 pour stockage dans Elasticsearch. Il définit le champ `@timestamp` utilisé pour l'indexation temporelle.

```
filter {
  date {
    match => [ "timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ]
    target => "@timestamp"
    timezone => "Europe/Paris"
  }
}
```

Le champ **match** définit le champ source et le format. **target** spécifie le champ destination (généralement @timestamp). **timezone** convertit vers le fuseau horaire désiré.

Plusieurs formats peuvent être testés séquentiellement. Le premier qui correspond est utilisé. En cas d'échec, tag _dateparsefailure est ajouté.

```
filter {
  date {
    match => [
      "timestamp",
      "ISO8601",
      "dd/MMM/yyyy:HH:mm:ss Z",
      "UNIX"
    ]
  }
}
```

En production, toujours définir timezone explicitement pour éviter ambiguïtés. Valider que @timestamp correspond au moment réel de l'événement.

Mutate : manipulation des champs

Suppression de champs

```
filter {
  mutate {
    remove_field => ["temporary", "debug_info"]
  }
}
```

Nettoie les champs inutiles avant indexation. Réduit la taille des documents et simplifie le mapping Elasticsearch.

Renommage

```
filter {
  mutate {
    rename => { "old_field" => "new_field" }
  }
}
```

Normalise les noms de champs entre différentes sources. Facilite requêtes et visualisations uniformes dans Kibana.

Conversion de type

```
filter {
  mutate {
    convert => {
      "response_time" => "integer"
      "bytes" => "float"
    }
  }
}
```

Force le type de données pour mapping correct. Évite erreurs de mapping conflicts dans Elasticsearch.

Ajout de champs

```
filter {
  mutate {
    add_field => {
      "environment" => "production"
      "datacenter" => "eu-west-1"
    }
  }
}
```

Enrichit événements avec métadonnées contextuelles. Utile pour filtrage multi-environnement.

GeoIP : enrichissement géographique

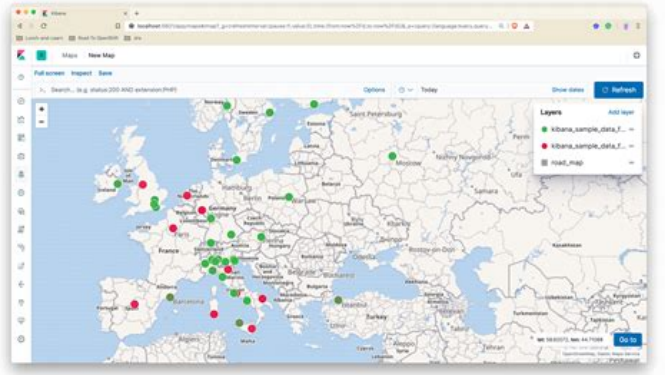
Le filtre GeoIP ajoute des informations de géolocalisation à partir d'adresses IP. Il utilise la base de données MaxMind GeoLite2 pour résoudre pays, ville, coordonnées GPS.

```
filter {
  geoip {
    source => "client_ip"
    target => "geoip"
    fields => ["city_name", "country_name", "location"]
  }
}
```

Le champ **source** contient l'IP à résoudre. **target** définit l'objet JSON de destination. **fields** limite les données extraites.

Résultat dans Elasticsearch :

```
{
  "client_ip": "8.8.8.8",
  "geoip": {
    "city_name": "Mountain View",
    "country_name": "United States",
    "location": {
      "lat": 37.386,
      "lon": -122.0838
    }
  }
}
```



Ces données permettent visualisations cartographiques dans Kibana. Mapping automatique du champ location en type geo_point.

Mise à jour de la base

Base GeoLite2 incluse avec Logstash. Mise à jour recommandée mensuelle via compte MaxMind gratuit.

Configuration dans logstash.yml :

```
xpack.geoip.downloader.enabled: true
```

Json et KV : parsing automatique

Filtre json

Parse une chaîne JSON en champs structurés. Extrait automatiquement tous les attributs du JSON.

```
filter {
  json {
    source => "message"
    target => "parsed"
  }
}
```

Si message contient `{"user":"alice","status":"ok"}`, le résultat sera :

```
{
  "parsed": {
    "user": "alice",
    "status": "ok"
  }
}
```

Sans target, les champs sont extraits à la racine de l'événement. Utile pour logs applicatifs déjà en JSON.

Filtre kv

Parse des paires clé=valeur séparées par espaces. Commun dans logs systèmes et applicatifs.

```
filter {
  kv {
    source => "message"
    field_split => " "
    value_split => "="
  }
}
```

Transforme `user=alice status=ok code=200` en :

```
{
  "user": "alice",
  "status": "ok",
  "code": "200"
}
```

Paramètres field_split et value_split personnalisables selon format source.

Fingerprint : génération d'identifiants

Le filtre fingerprint crée un identifiant unique basé sur le contenu d'un ou plusieurs champs. Utile pour déduplication ou définition d'un ID document personnalisé.

```
filter {
  fingerprint {
    source => ["client_ip", "timestamp", "message"]
    target => "[@metadata][fingerprint]"
    method => "SHA256"
    key => "my_secret_key"
  }
}

output {
  elasticsearch {
    hosts => ["https://es01:9200"]
    document_id => "%{[@metadata][fingerprint]}"
  }
}
```

Combine les champs client_ip, timestamp, message pour générer un hash SHA256. Stockage dans @metadata (non indexé). Utilisation comme document_id dans output.

Avantage : deux événements identiques produisent le même ID. Elasticsearch remplace document existant au lieu d'en créer un duplicata. Méthodes disponibles : MD5, SHA1, SHA256, SHA512, MURMUR3.

Bonnes pratiques pour les filtres

01

Ordre d'exécution

L'ordre des filtres est critique. Traiter d'abord grok pour extraction, puis date pour timestamp, puis enrichissements (geoip), enfin nettoyage (mutate remove_field).

02

Conditionnels

Utiliser des conditions pour appliquer filtres sélectivement selon type de log ou valeur de champ. Évite traitement inutile et améliore performance.

03

Gestion des erreurs

Grok ajoute tag _grokparsefailure en cas d'échec. Surveiller ces tags dans monitoring. Ajuster patterns si taux d'échec élevé.

04

Performance regex

Éviter patterns grok trop larges comme %{GREEDYDATA} en début. Privilégier patterns spécifiques. Tester charge avec vraies données avant production.

```
filter {
  if [type] == "apache" {
    grok {
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
  }
  if "_grokparsefailure" in [tags] {
    drop { }
  }
}
```

Exemple de conditionnel par type et suppression des événements non parsés.

Pipeline Apache complet avec enrichissement

```
input {
  beats {
    port => 5044
  }
}

filter {
  # Parsing des logs Apache
  grok {
    match => { "message" => "%{COMBINEDAPACHELOG}" }
  }

  # Normalisation du timestamp
  date {
    match => [ "timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ]
    target => "@timestamp"
  }

  # Conversion types numériques
  mutate {
    convert => {
      "response" => "integer"
      "bytes" => "integer"
    }
  }

  # Enrichissement géographique
  geoip {
    source => "clientip"
    target => "geoip"
  }

  # Extraction user-agent
  useragent {
    source => "agent"
    target => "user_agent"
  }

  # Nettoyage
  mutate {
    remove_field => ["message", "timestamp"]
  }
}

output {
  elasticsearch {
    hosts => ["https://es01:9200"]
    user => "logstash_writer"
    password => "${LOGSTASH_ES_PASSWORD}"
    ssl => true
    cacert => "/usr/share/logstash/config/certs/ca.crt"
    index => "apache-logs-%{+YYYY.MM.dd}"
  }
}
```

Pipeline complet avec parsing, normalisation temporelle, conversions de types, enrichissement géographique et extraction du user-agent. Champs inutiles supprimés avant indexation.

Outputs : destinations des données

Elasticsearch

Destination principale. Indexation avec gestion SSL, authentification, retry automatique. Support ILM et data streams.

File / stdout

Écriture fichier local ou console. Idéal pour debug et validation pipeline sans Elasticsearch.

Kafka / Redis

Redistribution vers systèmes de queue. Architecture distribuée avec découplage. Permet fan-out vers multiples consommateurs.

HTTP / Email

Intégrations externes. Webhooks, alerting email, APIs tierces. Notifications basées sur contenu événements.

Output Elasticsearch : paramètres critiques

```
output {
  elasticsearch {
    hosts => ["https://es01:9200", "https://es02:9200"]
    user => "logstash_writer"
    password => "${LOGSTASH_ES_PASSWORD}"
    ssl => true
    ssl_certificate_verification => true
    cacert => "/usr/share/logstash/config/certs/ca.crt"
    index => "logs-%{+YYYY.MM.dd}"
    document_id => "%{[@metadata][fingerprint]}"
    ilm_enabled => true
    ilm_rollover_alias => "logs"
    ilm_pattern => "{now/d}-000001"
    action => "create"
  }
}
```

Paramètres essentiels

- **hosts** : liste des nœuds Elasticsearch (load-balancing)
- **ssl** : activation HTTPS obligatoire en 9
- **cacert** : certificat CA pour validation serveur
- **index** : pattern d'index avec rotation temporelle
- **document_id** : ID personnalisé (déduplication)

Gestion des erreurs

Retry automatique avec backoff exponentiel. Dead letter queue (DLQ) pour événements en échec persistant.

```
dead_letter_queue.enable: true
path.dead_letter_queue: /var/lib/logstash/dlq
```

Surveillance des erreurs via API monitoring. Rejouer DLQ après correction problème.

Outputs pour debug et test

stdout

Affiche événements dans console. Codec rubydebug pour format lisible. Indispensable en phase développement.

```
output {
  stdout {
    codec => rubydebug
  }
}
```

Lancer Logstash en foreground avec `--debug` pour observer traitement temps réel.

file

Écrit événements dans fichier local. Rotation automatique possible. Utile pour archivage ou debug offline.

```
output {
  file {
    path => "/var/log/logstash/output.json"
    codec => json_lines
  }
}
```

Fichier JSON Lines : un événement par ligne. Facilite réingestion ou analyse avec jq.

Monitoring avec X-Pack

Logstash intègre des capacités de monitoring via X-Pack. Les métriques sont collectées et indexées dans Elasticsearch pour visualisation dans Kibana.

```
# Dans logstash.yml
xpack.monitoring.enabled: true
xpack.monitoring.elasticsearch.hosts: ["https://es01:9200"]
xpack.monitoring.elasticsearch.username: "logstash_monitoring"
xpack.monitoring.elasticsearch.password: "${MONITORING_PASSWORD}"
xpack.monitoring.elasticsearch.ssl.certificate_authority: "/path/to/ca.crt"
```

Métriques collectées : throughput (événements/sec), temps traitement par filtre, mémoire JVM, CPU, état de la queue, erreurs pipeline.

Dashboards prédéfinis dans Kibana : Stack Monitoring → Logstash. Vue temps réel de performance et santé.

API de monitoring Logstash

Logstash expose une API HTTP sur le port 9600 pour interrogation des statistiques et configuration.

```
# Stats générales
curl http://localhost:9600/_node/stats

# Stats pipeline spécifique
curl http://localhost:9600/_node/stats/pipelines/main

# Informations système
curl http://localhost:9600/_node/info

# Hot threads (debug performance)
curl http://localhost:9600/_node/hot_threads
```

Réponse JSON exploitable par scripts monitoring ou intégration Prometheus.

Métriques clés

- **events.in** : événements reçus
- **events.filtered** : traités avec succès
- **events.out** : envoyés en output
- **events.duration_in_millis** : temps traitement
- **queue.events** : taille buffer actuelle

Surveiller ratio events.out / events.in. Si inférieur à 1, perte ou filtrage excessif. Monitorer queue.events pour détecter saturation.

Logs internes et dashboards Kibana

Logs Logstash écrits dans `/var/log/logstash/logstash-plain.log`. Format texte avec niveau, timestamp, message. Niveaux : TRACE, DEBUG, INFO, WARN, ERROR, FATAL.

```
# Surveillance en temps réel
tail -f /var/log/logstash/logstash-plain.log

# Filtrage erreurs
grep ERROR /var/log/logstash/logstash-plain.log
```

Configuration niveau dans logstash.yml :

```
log.level: info
path.logs: /var/log/logstash
```

Dashboards Kibana dédiés disponibles après activation monitoring. Navigation : Stack Monitoring → Logstash. Visualisation graphique throughput, latences, erreurs par pipeline.

Alerting configurable sur métriques critiques : queue saturée, taux erreur élevé, CPU/RAM. Intégration avec Elasticsearch Watcher pour notifications.

Sécurité : connexion avec certificats

Elasticsearch 9 impose HTTPS et authentification. Logstash requiert certificats PEM pour validation SSL/TLS.

Génération des certificats

```
# Avec elasticsearch-certutil
bin/elasticsearch-certutil ca --pem
bin/elasticsearch-certutil cert --ca-cert ca/ca.crt --ca-key
ca/ca.key --pem
```

Résultat : ca.crt (autorité), instance.crt et instance.key (certificat nœud).

Copier ca.crt vers serveur Logstash dans `/usr/share/logstash/config/certs/`.

Configuration output

```
output {
  elasticsearch {
    hosts => ["https://es01:9200"]
    ssl => true
    ssl_certificate_verification => true
    cacert => "/usr/share/logstash/config/certs/ca.crt"
    user => "logstash_writer"
    password => "${LOGSTASH_ES_PASSWORD}"
  }
}
```

Variable d'environnement LOGSTASH_ES_PASSWORD définie dans systemd ou docker-compose. Évite stockage mot de passe en clair.

Utilisateur et rôle dédié Logstash

Créer utilisateur Elasticsearch spécifique pour Logstash avec privilèges minimaux. Principe du moindre privilège appliqué.

```
# Création du rôle
POST /_security/role/logstash_writer
{
  "cluster": ["monitor", "manage_ilm", "manage_index_templates"],
  "indices": [
    {
      "names": ["logs-*", "apache-logs-*"],
      "privileges": ["create_index", "write", "create", "auto_configure"]
    }
  ]
}

# Création de l'utilisateur
POST /_security/user/logstash_writer
{
  "password": "MotDePasseSecurise123!",
  "roles": ["logstash_writer"],
  "full_name": "Logstash Writer User"
}
```

Privilèges cluster : monitor (santé), manage_ilm (lifecycle), manage_index_templates (templates). Privilèges indices : create_index, write, create sur pattern logs-*. Auto_configure pour data streams.

Stocker password dans variable environnement. Rotation régulière recommandée.

Performance et tuning

4–8

pipeline.workers

Égal au nombre de CPU cores. 4 pour serveur 4 cores, 8 pour 8 cores. Ajuster selon charge observée.

125

pipeline.batch.size

Événements par lot. Valeur par défaut équilibrée. Augmenter si débit élevé et latence acceptable.

1gb

queue.max_bytes

Taille maximale queue persistante. 1 GB par défaut. Augmenter si pics d'ingestion importants.

Mémoire JVM

```
# Dans jvm.options
-Xms4g
-Xmx4g
```

Min et max identiques pour éviter resize. 25-30% RAM totale, maximum 8 GB. Au-delà, inefficace (GC overhead).

Optimisation Grok

Patterns spécifiques plutôt que génériques. Éviter ancres ^ et $ inutiles. Tester avec grok debugger. Surveiller tag _grokparsefailure.

Tests et validation du pipeline

01

Validation syntaxe

```
logstash --config.test_and_exit -f /etc/logstash/conf.d/pipeline.conf
```

Vérifie syntaxe configuration sans démarrer traitement. Détecte erreurs parsing avant déploiement.

02

Mode debug

```
logstash --debug -f /etc/logstash/conf.d/pipeline.conf
```

Logs verbeux avec détails traitement chaque événement. Observer filtres appliqués et transformations.

03

Vérification API

```
curl http://localhost:9600/_node/pipelines
```

État pipelines actifs, nombre workers, plugins chargés. Confirme pipeline opérationnel.

04

Validation Kibana

Management → Index Patterns. Vérifier index créés avec mapping correct. Dev Tools → rechercher documents indexés.

Déploiement avec Docker Compose

```
services:
  logstash:
    image: docker.elastic.co/logstash/logstash:9.1.5
    container_name: logstash
    volumes:
      - ./pipeline:/usr/share/logstash/pipeline:ro
      - ./config/logstash.yml:/usr/share/logstash/config/logstash.yml:ro
      - ./certs:/usr/share/logstash/config/certs:ro
    environment:
      - LOGSTASH_ES_PASSWORD=${LOGSTASH_ES_PASSWORD}
      - LS_JAVA_OPTS=-Xms2g -Xmx2g
    ports:
      - "5044:5044"
      - "9600:9600"
    networks:
      - elastic
    depends_on:
      - elasticsearch
    restart: unless-stopped

networks:
  elastic:
    driver: bridge
```

Volumes : pipeline (configs), logstash.yml (settings), certs (SSL). Variable environnement pour password. Dépendance Elasticsearch pour ordre démarrage.

Cas pratiques d'utilisation

Logs web Nginx/Apache

Parsing logs accès HTTP. Extraction IP, URL, status codes, user-agents. Enrichissement GeoIP pour cartographie trafic. Visualisation dashboards web analytics.

Audit système

Collecte syslog depuis serveurs Linux. Normalisation événements auth, sudo, firewall. Détection patterns suspects. Alerting sur tentatives accès non autorisées.

Données applicatives JSON

Logs applicatifs structurés déjà en JSON. Parsing direct sans Grok. Enrichissement avec métadonnées environnement. Corrélation avec traces APM.

Corrélation multi-sources

Agrégation logs application, base de données, load-balancer. Tags par source. Recherches cross-source par trace ID ou session. Investigation incidents globale.

Maintenance et montée de version

Migration 7.x → 9.x

Changements majeurs entre versions. Sécurité activée par défaut en 9.x. Révision authentification output. Mise à jour plugins incompatibles.

```
# Lister plugins installés
bin/logstash-plugin list

# Mettre à jour tous plugins
bin/logstash-plugin update

# Installer plugin spécifique
bin/logstash-plugin install logstash-filter-geoip
```

Tester configuration avant production avec `--config.test_and_exit`. Valider compatibilité patterns Grok.

Vérification compatibilité

Consulter release notes Elastic pour breaking changes. Plugins obsolètes listés dans documentation. Alternatives suggérées.

Plugins core maintenus par Elastic. Plugins communautaires peuvent nécessiter fork ou remplacement.

Bonnes pratiques

- Environnement staging pour tests
- Backup configurations avant migration
- Montée progressive (7.17 → 8.x → 9.x)
- Monitoring post-migration intensif