Module 1 : Les Fondamentaux de Logstash

Introduction

Logstash est un moteur de traitement de données qui collecte, transforme et envoie des logs vers une destination. Son fonctionnement repose sur un concept simple : INPUT → FILTER → OUTPUT.

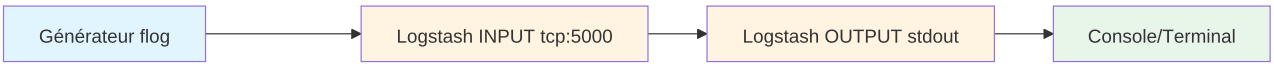
Dans ce premier module, nous allons commencer par le plus simple : recevoir des données et les afficher. Pas de filtres complexes, pas d'enrichissement, juste comprendre comment les données circulent dans Logstash.

Exercice 1.1 : Pipeline INPUT → OUTPUT

Objectif

Créer votre première pipeline Logstash qui reçoit des logs et les affiche dans la console.

Schéma de l'architecture



Concept clé

Une pipeline Logstash est un fichier de configuration qui définit trois sections :

- **input** : d'où viennent les données (fichiers, réseau, bases de données...)
- **filter** : comment transformer les données (optionnel pour le moment)
- **output** : où envoyer les données (fichiers, Elasticsearch, console...)

La structure de base ressemble à ceci :

```
input {
  # Configuration de la source
}

filter {
  # Transformations (optionnel)
}
```

```
output {
  # Configuration de la destination
}
```

Pratique guidée

L'infrastructure Docker est déjà en place. Vous allez travailler uniquement sur les fichiers de configuration Logstash.

Créez le fichier `logstash/pipeline/01-simple.conf` :

```
# Pipeline la plus simple possible
# INPUT : écoute sur le port TCP 5000
input {
  tcp {
    port => 5000                # Port d'écoute
    codec => json               # Interprète chaque message comme un objet JSON
    tags => ["simple_tcp"]      # Ajoute un tag à chaque événement pour filtrage
éventuel

    # SSL désactivé : transmission en clair
    ssl_enabled => false
    ssl_client_authentication => "none"  # Ignoré tant que SSL est désactivé
    ssl_verification_mode => "none"      # Inutile ici

    # Exemple de configuration SSL si activé
    # ssl_certificate_authorities => ["/usr/share/logstash/config/certs/ca/ca.crt"]
  }
}

# OUTPUT : affichage dans la console
output {
  stdout {
    codec => rubydebug          # Affiche les événements dans un format structuré
lisible
  }
}
```

Le service `flog` génère déjà des logs JSON automatiquement vers le port 5000. Vous pouvez aussi tester manuellement depuis le conteneur flog :

```
# Test manuel : envoyer un log JSON
docker compose exec -ti flog sh
echo '{"message": "Bonjour Logstash", "level": "info"}' | nc logstash 5000
```

Regardez les logs de Logstash pour voir le résultat :

```
docker compose logs -f --tail=100 logstash
```

Vous devriez voir votre message s'afficher avec tous les champs ajoutés automatiquement par Logstash (@timestamp, @version, host, etc.).

Option	Valeur	Statut et Impact
`ssl_enabled`	`false`	**Désactivé** - Cet input acceptera les données sur le port 5000 en texte clair (non chiffré). **Risque de sécurité majeur en production.**
`ssl_client_authentication`	`"none"`	**Désactivé** - Logstash n'exigera pas de certificat client. C'est la configuration attendue lorsque `ssl_enabled` est à `false`.
`ssl_verification_mode`	`"none"`	**Sans effet** - Cette option est ignorée lorsque `ssl_enabled` est `false`.

Comprendre la sortie

Quand Logstash reçoit votre message, il l'affiche avec ce format :

```
{
       "message" => "Bonjour Logstash",
         "level" => "info",
    "@timestamp" => 2025-10-07T14:30:15.123Z,
      "@version" => "1",
          "host" => "172.18.0.5",
          "port" => 45678,
          "tags" => [
        [0] "simple_tcp"
    ]
}
```

Logstash a automatiquement ajouté :

- `@timestamp` : quand le log a été reçu
- `@version` : version du schéma Logstash
- `host` et `port` : informations sur la connexion TCP
- `tags` : le tag qu'on a défini dans l'input

TP

Maintenant, modifiez votre pipeline pour :

1. Changer le port d'écoute de 5000 à 5001
2. Ajouter un deuxième tag "exercice_1"

3. Tester avec votre nouvelle configuration

Modifiez le fichier `logstash/pipeline/01-simple.conf` puis testez :

```
# Tester sur le nouveau port
echo '{"test": "nouveau port"}' | nc localhost 5001

# Voir le résultat
docker compose logs logstash | tail -20
```

Questions

Question 1

Quel est le rôle du paramètre `codec => json` dans l'input TCP ?

a) Il force l'envoi des données en JSON
b) Il indique à Logstash que les données entrantes sont au format JSON
c) Il convertit automatiquement tout texte en JSON
d) Il valide que le JSON est bien formé

Question 2

Si vous supprimez le bloc `output`, que se passe-t-il ?

a) Logstash refuse de démarrer
b) Les données sont perdues
c) Les données sont automatiquement envoyées vers Elasticsearch
d) Les données sont stockées en mémoire

Question 3

Comment pouvez-vous voir les logs générés par le service `flog` qui sont envoyés vers Logstash ?

a) `docker compose logs flog`
b) `docker compose logs logstash`
c) Les deux réponses sont correctes
d) Aucune des deux

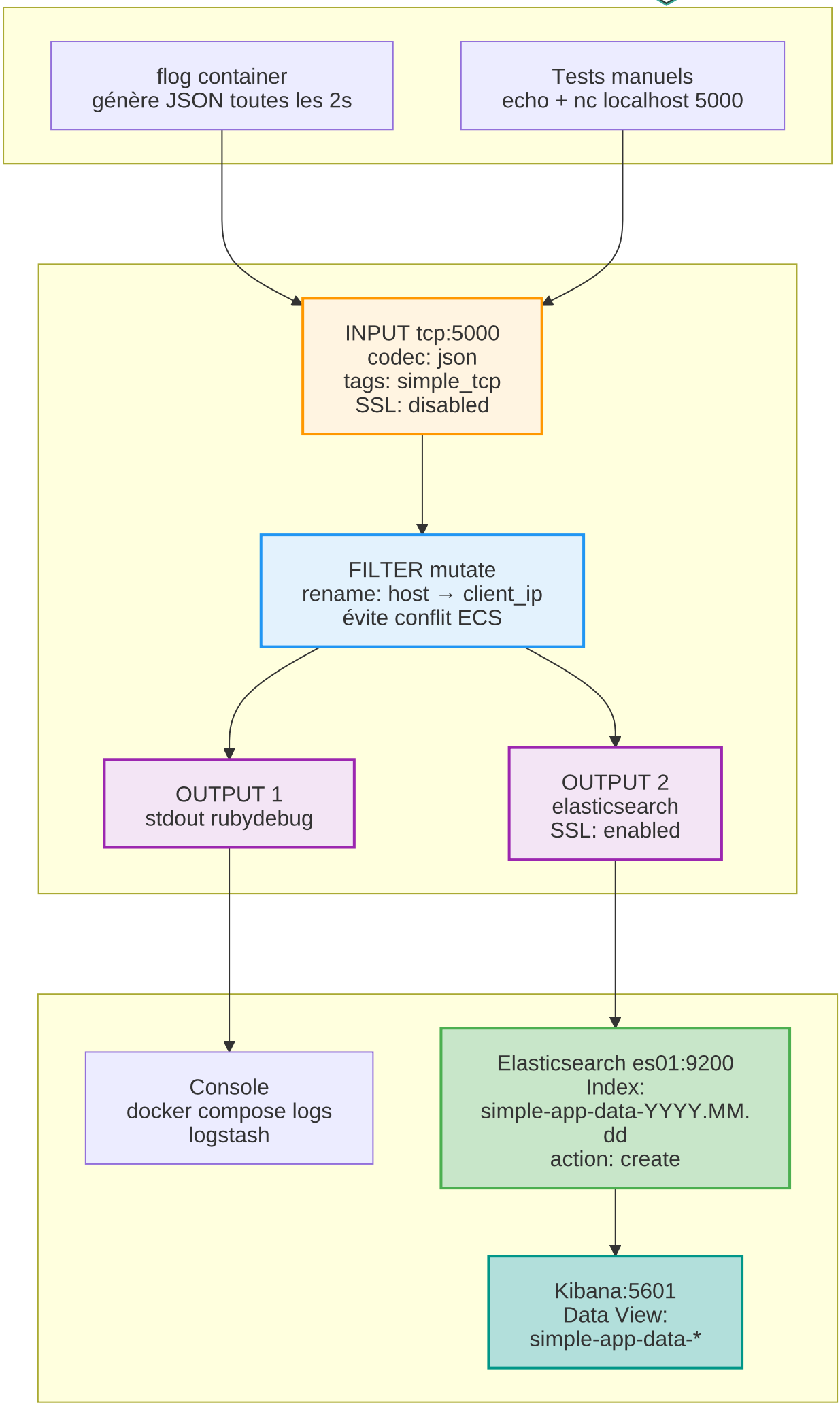
Exercice 1.2 : Connexion à Elasticsearch

Durée : 15 minutes

Objectif

Envoyer vos logs vers Elasticsearch au lieu de la console, et les visualiser dans Kibana.

Schéma de l'architecture


flog container
génère JSON toutes les 2s
Tests manuels
echo + nc localhost 5000
INPUT tcp:5000
codec: json
tags: simple_tcp
SSL: disabled
FILTER mutate
rename: host → client_ip
évite conflit ECS
OUTPUT 1
stdout rubydebug
OUTPUT 2
elasticsearch
SSL: enabled
Console
docker compose logs
logstash
Elasticsearch es01:9200
Index:
simple-app-data-YYYY.MM.
dd
action: create
Kibana:5601
Data View:
simple-app-data-*

Rappel

Elasticsearch est une base de données orientée documents qui stocke les logs sous forme d'index. Chaque log devient un document JSON dans un index.

Logstash peut envoyer des données vers Elasticsearch via l'output `elasticsearch`. La connexion nécessite :

- L'URL du cluster Elasticsearch
- Des identifiants (user/password)
- La configuration SSL si activée

Pratique guidée

Créez le fichier `logstash/pipeline/02-elasticsearch.conf` :

```
# Pipeline la plus simple possible
# INPUT : on écoute sur le port TCP 5000

input {
  tcp {
    port => 5000
    codec => json  # On s'attend à recevoir du JSON
    tags => ["simple_tcp"]

    # SSL désactivé pour ce input TCP
    ssl_enabled => false
    ssl_client_authentication => "none"
    ssl_verification_mode => "none"

    # Si la sécurité est requise, décommenter et configurer :
    # ssl_certificate_authorities => ["/usr/share/logstash/config/certs/ca/ca.crt"]
  }
}

filter {
  # Renomme le champ 'host' (qui contient l'IP du client) en 'client_ip'
  # pour éviter le conflit avec le mappage 'host' de l'ECS/Data Stream.
  mutate {
    rename => { "host" => "client_ip" }
  }
}


output {
  # OUTPUT : on affiche dans la console
  stdout {
    codec => rubydebug  # Format lisible pour debug
  }

  # On ajoute Elasticsearch
  elasticsearch {
```

```
    hosts => ["https://es01:9200"]
    user => "elastic"
    password => "${ELASTIC_PASSWORD}"

    # Configuration SSL
    ssl_enabled => true
    ssl_certificate_authorities => ["/usr/share/logstash/config/certs/ca/ca.crt"]

    # force la création de l'index
    action => "create"
    # Nom de l'index
    index => "simple-app-data-%{+YYYY.MM.dd}"
  }
}
```

Vérification dans Kibana

Ouvrez votre navigateur et allez sur : https://localhost:5601

Connectez-vous avec :

- Username : `elastic`
- Password : `changeme`

Une fois connecté :

- Ouvrir le **menu latéral** (≡).
- Aller dans **Analytics** → **Discover**.
- En haut, cliquer sur le nom du **data view** (souvent “Default”).
- Sélectionner **Manage data views**. Choisir **All sources**
- Cliquer sur Edit data view
- Remplir :
  - **Name** : `simple-app-data-*`
  - **Index pattern** : `simple-app-data-*`
  - **Timestamp field** : `@timestamp`
- Cliquer sur **Save en haut à droite, choisir un nom et continue to use (close en bas).**

Vous devriez voir vos logs apparaître avec tous leurs champs. Explorez l'interface :

- Cliquez sur un document pour voir tous ses champs
- Utilisez la barre de recherche pour filtrer
- Changez la période temporelle en haut à droite

Comprendre l'index pattern

Dans la configuration, nous avons utilisé :

```
index => "simple-app-data-%{+YYYY.MM.dd}"
```

Cela signifie que Logstash crée un index par jour :

- `simple-app-data-2025.10.27` pour aujourd'hui
- `simple-app-data-2025.10.28` pour demain
- etc.

Le pattern `simple-app-data-*` dans Kibana permet de rechercher dans tous ces index en même temps.

TP

Modifiez votre pipeline pour :

1. Changer le nom de l'index en `test-logs-%{+YYYY.MM.dd}`
2. Créer un nouveau Data View dans Kibana avec le pattern `test-logs-*`
3. Envoyer 3 nouveaux logs et les visualiser

```
# Envoyer des logs de test
echo '{"message": "Test nouveau index", "status": "ok"}' | nc localhost 5000
```

Questions

Question 1

Pourquoi utilise-t-on `${ELASTIC_PASSWORD}` au lieu d'écrire le mot de passe directement ?

a) C'est plus court à écrire
b) Pour des raisons de sécurité, le mot de passe est stocké dans les variables d'environnement
c) Logstash n'accepte pas les mots de passe en dur
d) C'est obligatoire pour la connexion SSL

Question 2

Que se passe-t-il si vous ne configurez pas `ssl_enabled => true` alors qu'Elasticsearch utilise HTTPS ?

a) La connexion fonctionne sans problème
b) Logstash refuse de démarrer
c) Logstash ne peut pas se connecter et affiche des erreurs SSL
d) Les données sont envoyées en clair

Question 3

Quel avantage y a-t-il à créer un index par jour avec `%{+YYYY.MM.dd}` ?

a) C'est plus joli visuellement
b) Ça permet de gérer la rétention des données plus facilement (supprimer les vieux index)
c) Ça accélère les recherches sur des périodes courtes
d) Les réponses b et c sont correctes

Conceptes

Concepts maîtrisés

- Structure d'une pipeline : input → filter → output
- Configuration d'un input TCP avec codec JSON
- Output vers stdout pour le debug
- Output vers Elasticsearch avec SSL
- Index patterns et rotation par date
- Variables d'environnement pour les secrets

Commandes utiles

```
# Voir les logs de Logstash
docker compose logs -f logstash

# Envoyer un log de test
echo '{"test": "value"}' | nc localhost 5000

# Voir les derniers logs traités
docker compose logs logstash | tail -20
```

Fichiers créés

- `logstash/pipeline/01-simple.conf` : pipeline basique stdout
- `logstash/pipeline/02-elasticsearch.conf` : pipeline avec Elasticsearch