**Introduction to Ingest Pipelines**

Ingest pipelines in Elastic Search allow you to preprocess documents before indexing them. They are particularly useful when working with raw log data that needs transformation, enrichment, or cleanup.

**Why Use Ingest Pipelines?**

* **Data Transformation:** Modify fields, extract values, and clean up log entries.
* **Log Enrichment:** Add metadata like geo-location, user agent, or timestamp formatting.
* **Efficient Parsing:** Preprocess logs before they are stored in Elastic Search, reducing query-time processing.

**Components of an Ingest Pipeline**

An ingest pipeline consists of a series of **processors** that manipulate data before it is indexed. Common processors include:

* **Grok Processor:** Pattern matching for extracting structured data from unstructured logs.
* **Date Processor:** Converts date strings into proper timestamps.
* **Geo IP Processor:** Enriches logs with geographical information.
* **Rename Processor:** Renames fields for better consistency.

**Setting Up an Ingest Pipeline**

**Step 1: Create an Ingest Pipeline**

You can define an ingest pipeline using the Elastic Search API.

PUT \_ingest/pipeline/my\_pipeline

{

"description": "Parse and structure logs",

"processors": [

{

"grok": {

"field": "message",

"patterns": ["%{timestamp} %{+timestamp} %{IP} %{Method} %{URI} user=%{user} %{port} %{admin} %{client\_IP} %{browser} %{responce\_code} %{uri} %{uri2} %{number}"]

}

},

{

"date": {

"field": "timestamp",

"formats": ["dd/MMM/yyyy:HH:mm:ss Z"]

}

}

]

}

**Step 2: Configure Filebeat to Use the Pipeline**

Modify your **filebeat.yml** configuration to send logs through the pipeline:

filebeat.inputs:

- type: log

enabled: true

paths:

- /var/log/apache2/access.log

pipeline: "my\_pipeline"

**Step 3: Verify Data in Elastic Search**

After Filebeat sends logs, verify the processed output:

GET filebeat-\*/\_search

{

"query": {

"match\_all": {}

}

}

POST \_ingest/pipeline/apache\_logs\_pipeline/\_simulate

{

"docs": [

{

"\_source": {

"message": "192.168.1.100 - - [27/Mar/2025:12:45:32 +0000] \"GET /index.html HTTP/1.1\" 200 5324"

}

}

]

}

### ****Scenario 3: Processing JSON Logs****

#### **Use Case:** Extract fields from JSON logs stored as a string.

**Sample Log Entry**

{

"message": "{\"user\": \"john\_doe\", \"action\": \"login\", \"status\": \"success\"}"

}

#### **Ingest Pipeline Definition**

* Use json processor to parse message into separate fields.

PUT \_ingest/pipeline/json\_logs\_pipeline

{

"description": "Pipeline to parse JSON logs",

"processors": [

{

"json": {

"field": "message",

"target\_field": "log\_data"

}

},

{

"rename": {

"field": "log\_data.user",

"target\_field": "user"

}

},

{

"rename": {

"field": "log\_data.action",

"target\_field": "action"

}

},

{

"rename": {

"field": "log\_data.status",

"target\_field": "status"

}

},

{

"remove": {

"field": ["message", "log\_data"]

}

}

]

}

#### **Testing the Pipeline**

POST \_ingest/pipeline/json\_logs\_pipeline/\_simulate

{

"docs": [

{

"\_source": {

"message": "{\"user\": \"john\_doe\", \"action\": \"login\", \"status\": \"success\"}"

}

}

]

}

### ****Scenario 4: Masking Sensitive Data****

#### **Use Case:** Remove or mask sensitive information (e.g., credit card numbers).

**Sample Log Entry**

User: John Doe, Credit Card: 4111-1111-1111-1111, Action: Purchase

#### **Ingest Pipeline Definition**

* Use gsub processor to mask credit card numbers.

PUT \_ingest/pipeline/mask\_sensitive\_data\_pipeline

{

"description": "Pipeline to mask credit card details",

"processors": [

{

"gsub": {

"field": "message",

"pattern": "\\b\\d{4}-\\d{4}-\\d{4}-\\d{4}\\b",

"replacement": "\*\*\*\*-\*\*\*\*-\*\*\*\*-\*\*\*\*"

}

}

]

}

#### **Testing the Pipeline**

POST \_ingest/pipeline/mask\_sensitive\_data\_pipeline/\_simulate

{

"docs": [

{

"\_source": {

"message": "User: John Doe, Credit Card: 4111-1111-1111-1111, Action: Purchase"

}

}

]

}

### ****Scenario 5: Automatically Categorizing Log Severity****

#### **Use Case:** Add a log\_level field based on keywords.

**Sample Log Entry**

[ERROR] Database connection failed.

#### **Ingest Pipeline Definition**

* Use set processor to assign a log\_level based on message content.

PUT \_ingest/pipeline/log\_severity\_pipeline

{

"description": "Pipeline to categorize logs by severity",

"processors": [

{

"grok": {

"field": "message",

"patterns": ["\\[%{WORD:log\_level}\\] %{GREEDYDATA:log\_message}"]

}

},

{

"lowercase": {

"field": "log\_level"

}

}

]

}

#### **Testing the Pipeline**

POST \_ingest/pipeline/log\_severity\_pipeline/\_simulate

{

"docs": [

{

"\_source": {

"message": "[ERROR] Database connection failed."

}

}

]

}

**Hands-On Exercise**

1. Install File beat and configure it to read sample logs.
2. Create an ingest pipeline in Elastic Search.
3. Modify Filebeat to send logs through the pipeline.

This guide provides a structured approach for students to learn about ingest pipelines in a practical way. You can modify the exercises to include IIS logs or other log sources based on your teaching needs.

1. Check transformed data in Kibana.

**Conclusion**

Using ingest pipelines with File beat simplifies log processing and enhances data quality before indexing. This approach makes Elastic Search more efficient for searching and analyzing log data.