Плагины фильтрации

Поскольку в следующей главе мы будем детально рассматривать различные способы изменения и дополнения данных логов с помощью разных плагинов фильтрации, сейчас мы пропустим связанную с этим информацию.

Узел поглощения данных

До версии Elasticsearch 5.0, если нам требовалась предварительная обработка документов до индексирования, единственным возможным способом сделать это было использовать Logstash или преобразовывать их вручную, а потом индексировать в Elasticsearch. Не хватало функционала для возможности предварительной обработки/трансформации документов, они индексировались в исходном виде. Однако в версии Elasticsearch 5.x и далее была представлена функция узла поглощения данных, которая является легковесным решением для предварительной обработки и дополнения документов в Elasticsearch до индексирования.

Если Elasticsearch запущена в конфигурации по умолчанию, она будет работать как главный узел, узел данных и узел поглощения. Для отключения функции поглощения измените следующую настройку в файле `elasticsearch.yml`:

```
node.ingest: false
```

Узел поглощения может быть использован для предварительной обработки документов до момента фактического индексирования. Эта обработка происходит путем перехвата составных и индексирующих запросов, применения трансформаций данных и отправки документов обратно к API. С появлением новой функции поглощения Elasticsearch переняла часть функционала Logstash, ответственную за фильтрацию, и теперь мы можем выполнять обработку сырых логов и дополнять их внутри Elasticsearch.

Для предварительной обработки логов перед индексированием необходимо указать контейнер (который выполняет последовательность действий по трансформации входящего документа). Для этого мы попросту указываем параметр `pipeline` в индексе или в составном запросе, чтобы узел поглощения знал, какой контейнер использовать:

```
POST my_index/my_type?pipeline=my_pipeline_id
{
  "key": "value"
}
```

Определение контейнера

Контейнер определяет серию процессоров. Каждый процессор трансформирует документ в том или ином виде. Каждый процессор запускается в том порядке, в котором он определен в контейнере. Контейнер хранит два основных поля:

описание и список процессоров. Параметр `description` — необязательное поле, которое предназначено для хранения информации, связанной с использованием контейнера; с помощью параметра `processors` можно указать список процессоров для трансформации документа.

Типичная структура контейнера выглядит следующим образом:

```
{
  "description" : "...",
  "processors" : [ ... ]
}
```

Узел поглощения имеет около 20 встроенных процессоров, включая gsub, grok, convert, remove, rename и пр. для использования в создании контейнеров. Вместе со встроенными процессорами вы также можете применять доступные плагины поглощения, такие как ingest attachment, ingest geo-ip, ingest user-agent. По умолчанию они недоступны, но можно установить их как любой другой плагин Elasticsearch.

Ingest API

Узел поглощения предоставляет набор API, известный как Ingest API. Вы можете использовать эти API для определения, симуляции, удаления или поиска информации о контейнерах. Конечная точка данного API — `_ingest`.

API определения контейнера

Этот API используется для определения и добавления нового контейнера, а также для обновления имеющихся контейнеров.

Рассмотрим пример. Как видно в следующем коде, мы определили новый контейнер с названием `firstpipeline`, который переводит значения поля `message` в верхний регистр:

```
curl -X PUT http://localhost:9200/_ingest/pipeline/firstpipeline -H
'content-type: application/json'
-d '{
  "description" : "uppercase the incoming value in the message field",
  "processors" : [
    {
      "uppercase" : {
        "field": "message"
      }
    }
  ]
}'
```

При создании контейнера вы можете указать несколько процессоров, а порядок их выполнения зависит от порядка, в котором они указаны в конфигурации. Рассмотрим это на примере. Как видно в следующем коде, мы создали новый контейнер с именем `secondpipeline`, который преобразует значения поля `"message"`

в верхний регистр и переименовывает поле "message" в "data". Он создает новое поле с названием "label" и значением testlabel:

```
curl -X PUT http://localhost:9200/_ingest/pipeline/secondpipeline -H
'content-type: application/json'
-d '{
  "description" : "uppercase the incomming value in the message field",
  "processors" : [
    {
      "uppercase" : {
        "field": "message",
        "ignore_failure" : true
      }
    },
    {
      "rename": {
        "field": "message",
        "target_field": "data",
        "ignore_failure" : true
      }
    },
    {
      "set": {
        "field": "label",
        "value": "testlabel",
        "override": false
      }
    }
  ]
}'
```

Используем второй контейнер для индексирования документа-образца:

```
curl -X PUT 'http://localhost:9200/myindex/mytpe/1?pipeline=secondpipeline'
H 'content-type: application/json' -d '{
  "message":"elk is awesome"
}'
```

А теперь мы получим тот же документ и утвердим трансформацию:

```
curl -X GET http://localhost:9200/myindex/mytpe/1 -H
'content-type: application/json'

Ответ:
{
  "_index": "myindex",
  "_type": "mytpe",
  "_id": "1",
  "_version": 1,
  "found": true,
  "_source": {
    "label": "testlabel",
    "data": "ELK IS AWESOME"
  }
}
```

Если отсутствует поле, используемое процессором, тогда процессор сгенерирует исключение и документ не будет проиндексирован. Для предотвращения исключений процессора мы можем присвоить параметру "ignore_failure" значение true.

API получения контейнера

Этот API используется для получения определения существующего контейнера. С его помощью можно найти детали определения одного контейнера или всех.

Команда для поиска определений всех контейнеров выглядит следующим образом:

```
curl -X GET http://localhost:9200/_ingest/pipeline -H
'content-type: application/json'
```

Для поиска определения существующего контейнера введите его идентификатор для получения `.api` контейнеров. В примере ниже вы можете увидеть поиск определения контейнера с названием `secondpipeline`:

```
curl -X GET http://localhost:9200/_ingest/pipeline/secondpipeline -H
'content-type: application/json'
```

API удаления контейнера

Данный API удаляет контейнеры согласно идентификатору или совпадению с универсальным символом. Далее вы увидите пример удаления контейнера с названием `firstpipeline`:

```
curl -X DELETE http://localhost:9200/_ingest/pipeline/firstpipeline -H
'content-type: application/json'
```

API симуляции контейнера

Используется для симуляции выполнения контейнера относительно набора документов, выбранных в теле запроса. Можно указать как существующий контейнер, так и определение необходимого контейнера. Для симуляции контейнера поглощения добавьте конечную точку `"_simulate"` к API контейнера.

Ниже приведен пример симуляции существующего контейнера:

```
curl -X POST
http://localhost:9200/_ingest/pipeline/firstpipeline/_simulate -H
'contenttype: application/json' -d '{
  "docs" : [
    { "_source": {"message":"first document"} },
    { "_source": {"message":"second document"} }
  ]
}'
```

Далее вы увидите пример симулированного запроса с определением контейнера в теле запроса:

```
curl -X POST http://localhost:9200/_ingest/pipeline/_simulate –H
'contenttype: application/json' -d '{
```

```
  "pipeline" : {
    "processors":[
      {
        "join": {
          "field": "message",
          "separator": "-"
        }
      }]
  },
  "docs" : [
    { "_source": {"message":["first","document"]} }
  ]
}'
```

Резюме

В этой главе мы поговорили об основах Logstash. Мы установли и настроили Logstash, чтобы можно было начать работу с контейнерами данных, а также разобрались в архитектуре Logstash.

Вы узнали об узле поглощения (ingest node), который впервые появился в версии Elastic 5.x и который сегодня можно использовать вместо выделенной установки Logstash. Вы научились применять узел поглощения для предварительной обработки документов до момента фактического индексирования, а также познакомились с различными API.

В следующей главе мы поговорим о разнообразных фильтрах в арсенале Logstash, которые ставят его в один ряд с другими фреймворками потоковой обработки в реальном и псевдореальном времени с нулевым кодированием.