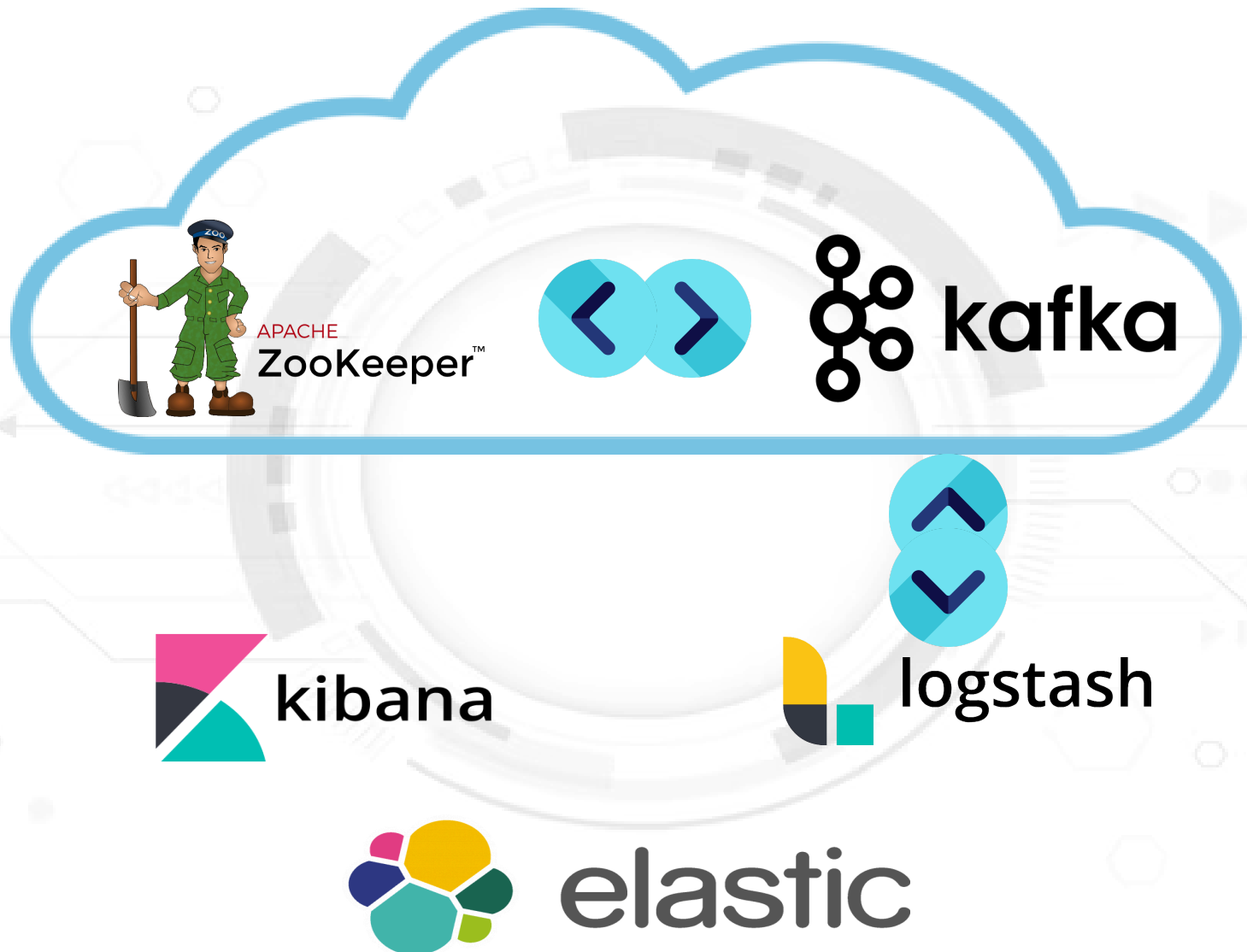


Proceso de ETL



Edvard Khachatryan Sahakyan

IES Abastos (Valencia, Valencia)
Big Data Aplicada
Vicent Tortosa
2023-2024

Índice

Introducción al proyecto	3
1. Recogida de los datos.	4
1.1 Recogida via CSV	4
1.2 Recogida via consumidor Kafka	4
1.2.1 Proceso realizado	4
Comando adicional	5
1.2.2 Automatización	5
1.2.3 Contratiempos	7
2. Archivo de configuración	8
2.1 Proceso del fichero CSV	8
2.1.1 Filtrado	8
2.2 Proceso del fichero JSON	10
3. Creación de Mapping	12
3.1 Procedimiento y toma de decisión	12
3.2 Contratiempo	13
4. Creación del “Index-Pattern”	15
5. Dashboard	17
Conclusión personal	21
Webgrafía	22

Introducción al proyecto

El proyecto comienza en octubre de 2023, siendo el primer objetivo seleccionar una web. Usando python y las debidas librerías, obtenemos el HTML el cual se analiza, para llegar a sacar una cantidad razonable de productos. Estos serán procesados en el mismo archivo “.py” para posteriormente generar un objeto de tipo diccionario y la colección completa escribirla en un “.csv” con los datos estructurados correctamente y la fecha del día de ejecución. Este algoritmo se ejecuta cada madrugada a las 03:00 AM.

Es importante mencionar que en el mismo servidor hay un zookeeper con un clusters de kafka, donde hay un topic personal para estos datos “*miBarbacoaData*”. Después de ejecutar el web scraping, este envía un mail desde el servidor a una cuenta de correo personal para confirmar cómo ha ido, con la información de artículos scrapeados, si ha habido algún error, e incluso se da el caso de que no se ha recibido el correo, se entiende que el servidor puede haber “caído”.

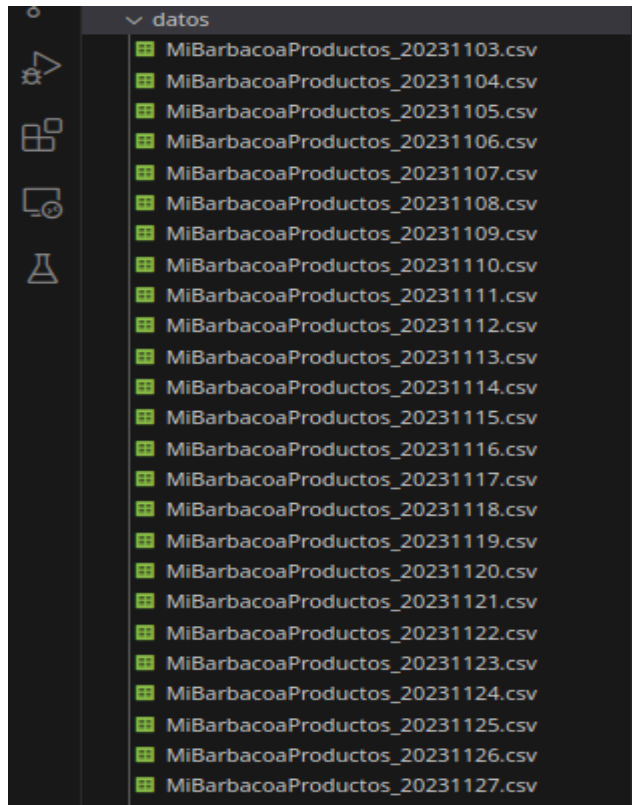
Después de seis meses, se alcanza el siguiente paso. Con las nuevas habilidades obtenidas con elasticsearch y sus componentes junto a Kibana, toca crear una ETL. El objetivo es consumir estos datos de dos maneras, “descargando los CSV de manera manual”, o “consumiendolo desde un *kafka consumer*”. El intermediario será Logstash, usando el archivo “.conf” indicaremos en su input, filter y output el algoritmo necesarios para cumplir con el objetivo del proyecto. Se crea un pipeline, dónde se puede indicar cuantos archivos de configuración se desee. Se muestra en este documento el proceso seguido para lograrlo.

Finalmente estos datos se quedarán registrados en “elasticsearch” y haciendo uso de “kibana” crearemos un par de “dashboards” para sacar las conclusiones de los productos que hemos escrapeado desde octubre de 2023 hasta el jueves 16 de mayo de 2024.

Adjunto enlace al repositorio en Github para consultar el código y recursos creados para el proyecto:

<https://github.com/EdvardKS/Specialization-BigdataApplied>

1. Recogida de los datos.



1.1 Recogida via CSV

En este punto se accede con VSCode y una extensión que es “connect ssh” para tener un acceso FTP.

Una vez en el servidor, desde el IDE se obtiene la opción para seleccionar los archivos deseados y descargar de manera manual la carpeta con los ficheros CSV.

En esta imagen se observa los documentos CSV en el servidor.

1.2 Recogida via consumidor Kafka

1.2.1 Proceso realizado

Para recordar el proceso cotidiano de crear un consumidor de un topic de Kafka, se accede y ejecuta en el servidor el siguiente comando para ver los resultados del topic.

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic miBarbacoaData --from-beginning
```

Ahora intentaremos que desde un archivo python consumir el mismo topic, y desde el mismo servidor.

El código lo estructuro de la siguiente manera:

```
from kafka import KafkaConsumer
consumer = KafkaConsumer('miBarbacoaData',
                          bootstrap_servers=['localhost:9092'],
                          auto_offset_reset='earliest',
                          enable_auto_commit=True,
                          group_id='my-group')
for message in consumer:
    print(message.value.decode('utf-8'))
```

El objetivo ahora es ejecutar el anterior algoritmo en python desde un ordenador externo que apunte al servidor dónde se encuentra este topic.

Para conseguirlo se forma el comando adecuado, en el ordenador externo para conectar con el servidor, indicando el comando interno que se ejecuta en el servidor el cual cargará el algoritmo antes mostrado.

El comando es el siguiente:

```
edvardunix@EdvardKS-OMEN:/mnt/c/Users/Edwar$ ssh -p 35001 8ia-edvard@abastos.duckdns.org 'python3 /home/8ia-edvard/bdAplic/consumidor_kafka.py'
```

Es importante indicar que se ejecuta desde el WSL del ordenador externo para que funcione. Ya que cmd no tiene esos comandos y genera conflictos al ejecutarlo, obteniendo el siguiente mensaje de error: "Command not found".

Una vez recibidos los datos, toca hacer un nuevo tipo de filtrado, ya que los datos se van a recibir de otra manera. Esto lo veremos en el punto ["2.2 Proceso de fichero JSON"](#)

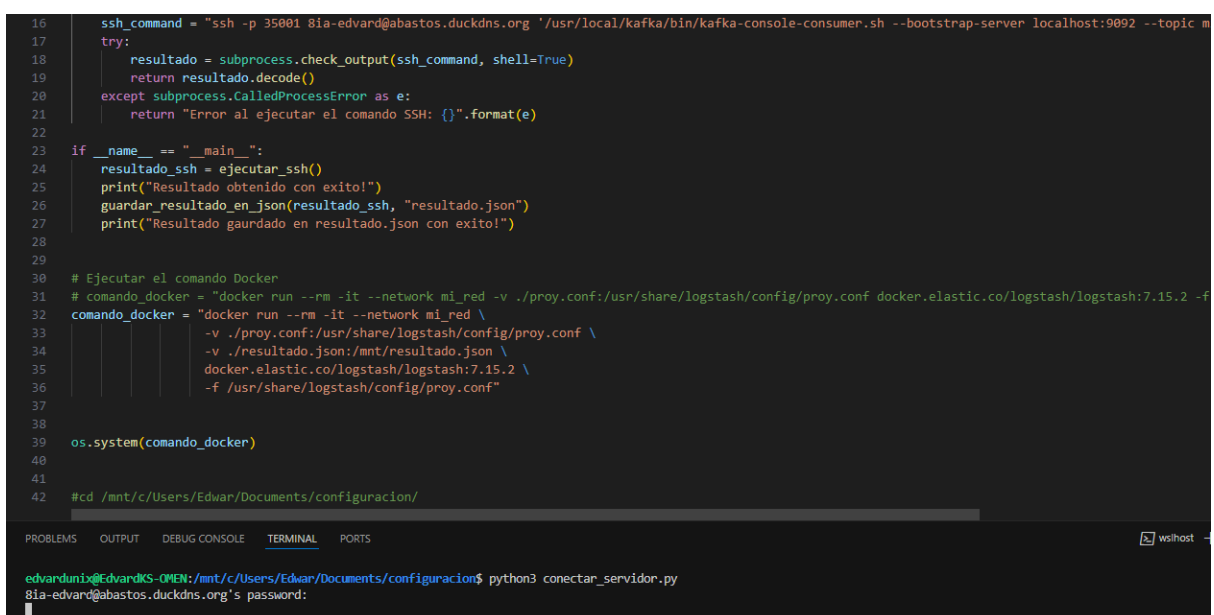
Comando adicional

Otra manera adicional para consumir el contenido del topic, es con el siguiente comando:

```
ssh -p 35001 8ia-edvard@abastos.duckdns.org '/usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic miBarbacoaData --from-beginning'
```

En concreto este comando ejecuta directamente un consumidor usando el servidor y creando un kafka consumer manualmente.

La desventaja es que no hay un control de la cantidad de líneas del topic que puede enviar el servidor, el cual dejaría "colgado" el servidor.



```
16  ssh_command = "ssh -p 35001 8ia-edvard@abastos.duckdns.org '/usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic m
17  try:
18      resultado = subprocess.check_output(ssh_command, shell=True)
19      return resultado.decode()
20  except subprocess.CalledProcessError as e:
21      return "Error al ejecutar el comando SSH: {}".format(e)
22
23  if __name__ == "__main__":
24      resultado_ssh = ejecutar_ssh()
25      print("Resultado obtenido con éxito!")
26      guardar_resultado_en_json(resultado_ssh, "resultado.json")
27      print("Resultado gaurdado en resultado.json con éxito!")
28
29
30  # Ejecutar el comando Docker
31  # comando_docker = "docker run --rm -it --network mi_red -v ./proy.conf:/usr/share/logstash/config/proy.conf docker.elastic.co/logstash/logstash:7.15.2 -f
32  comando_docker = "docker run --rm -it --network mi_red \
33  -v ./proy.conf:/usr/share/logstash/config/proy.conf \
34  -v ./resultado.json:/mnt/resultado.json \
35  docker.elastic.co/logstash/logstash:7.15.2 \
36  -f /usr/share/logstash/config/proy.conf"
37
38
39  os.system(comando_docker)
40
41
42  #cd /mnt/c/Users/Edwar/Documents/configuracion/
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
edvardunix@EdvardKS-OMEN:/mnt/c/Users/Edwar/Documents/configuracion$ python3 conectar_servidor.py
8ia-edvard@abastos.duckdns.org's password:
```

En esta imagen observamos cómo el servidor no es capaz de dar respuesta a esa cantidad de datos.

1.2.2 Automatización

Todo el proceso de recogida de datos desde kafka, se resume en un orden de comandos, el cual apoyado con docker, se logra automatizar todo el proceso. El orden de ejecución es el siguiente:

- Ejecución del comando que accede al servidor de kafka. Ejecuta el archivo python que crea un consumidor y devolverá por consola los topics.
- El ordenador externo recoge los datos en una variable tipo string, los cuales se pasarán a json object de python, después de hacer el debido tratamiento.
- Automáticamente se ejecuta el comando de arranque de logstash en un contenedor Docker. Se copia en sus debidas rutas los archivos de configuración modificados como, pipelines, yml, y conf. Posteriormente se hará la inyección en elasticsearch.

```

conectar_servidor.py > ...
3 import os
4
5 def guardar_resultado_en_json(resultado, nombre_archivo):
6     with open(nombre_archivo, 'w') as f:
7         json.dump(resultado, f)
8
9 def ejecutar_ssh():
10     # Este hace uso del archivo py del servidor para crear un consumidor y consumir el topic de kafka
11     ssh_command = "ssh -p 35001 8ia-edvard@abastos.duckdns.org 'python3 /home/8ia-edvard/bdAplic/consumidor_kafka.py'"
12     # Este crea un consumidor de forma nativa para atacar a la topic y consumir su contenido
13     # ssh_command = "ssh -p 35001 8ia-edvard@abastos.duckdns.org '/usr/local/kafka/bin/kafka-console-consumer.sh --bootstra
14     try:
15         resultado = subprocess.check_output(ssh_command, shell=True)
16         return resultado.decode()
17     except subprocess.CalledProcessError as e:
18         return "Error al ejecutar el comando SSH: {}".format(e)
19
20 if __name__ == "__main__":
21     resultado_ssh = ejecutar_ssh()
22     print("Resultado obtenido con exito!")
23     guardar_resultado_en_json(resultado_ssh, "resultado.csv")
24     print("Resultado gaurdado en resultado.json con exito!")
25
26
27 # Ejecutar el comando Docker
28 # comando_docker = "docker run --rm -it --network mi_red -v ./proy.conf:/usr/share/logstash/config/proy.conf docker.elastic
29 comando_docker = "docker run --rm -it --network elog \
30     -v ./pipelines.yml:/usr/share/logstash/config/pipelines.yml \
31     -v ./logstash.yml:/usr/share/logstash/config/logstash.yml \
32     -v ./proy.conf:/usr/share/logstash/config/proy.conf \
33     -v ./resultado.json:/home/resultado.json \
34     docker.elastic.co/logstash/logstash:7.17.21 \
35     -f /usr/share/logstash/config/proy.conf"
36
37
38 os.system(comando_docker)

```

Este es el archivo principal que automatizará todo el proceso de ETL.

Link a la carpeta dónde encontramos todos los archivos de configuración utilizados.

<https://github.com/EdvardKS/Specialization-BigdataApplied/tree/main/automatizacion>

1.2.3 Contratiempos

Un contratiempo ha sido el tema del grupo. Este no puede repetirse muchas veces o tiene algún límite. Para solucionarlo, se opta por modificar este nombre en cada ejecución. Con esto queda un registro de fecha y hora de ejecución del comando y creación de grupo:

```
nombre_grupo = 'grupo_' + datetime.now().strftime('%Y%m%d%H%M%S')
```

Otro contratiempo es que la cantidad de líneas en el topic es muy grande, por lo que el traspaso de esa cantidad de datos no está permitida por el servidor. Como ocupa mucha memoria, no es posible transportar esa cantidad de datos. Esto es debido a que por defecto este servidor está limitado y no está preparado para permitir pasar tanta información. Para solucionarlo he limitado el envío de los resultados a 100:

```
from kafka import KafkaConsumer
from datetime import datetime
nombre_grupo = 'grupo_' + datetime.now().strftime('%Y%m%d%H%M%S')
consumer = KafkaConsumer('miBarbacoaData',
                          bootstrap_servers=['localhost:9092'],
                          auto_offset_reset='earliest',
                          enable_auto_commit=True,
                          group_id=nombre_grupo)

variable = []
i = 0
for message in consumer:
    variable.append(message.value.decode('utf-8'))
    if i == 100:
        break
    else:
        i+=1
print(variable)
```

Se observa el código en el servidor, crea el consumidor, se recorre los datos del topic, pero con el límite de 100.

Otra dificultad a superar, ha sido el creer que no funcionaba el contenedor efímero que crea el algoritmo en cada ejecución para arrancar logstash. Se llegó a pensar que no cogía el archivo de configuración, y por ende, este archivo de configuración no se ejecutaba. Pero el error estaba en el formato del json. Sucede que logstash no es capaz de leer el JSON, si este no está bien formado, entonces no sigue la ejecución del archivo "proy.conf".

Se soluciona añadiendo un paso intermedio en la recogida de datos, el cual python formatea el json a un formato correcto, y luego monta el contenedor de logstash para hacer el filtrado e inyección en elasticsearch.

```
"[\"{'titulo' ... 'disponible': True}]\n"
```

Este sería un ejemplo de código mal formado.

2. Archivo de configuración

En este paso se crea el archivo de configuración llamado "proy.conf" este es el archivo que ejecutará Logstash para consumir los datos que se introducen en el input.

Hará un filtrado y modificado de los datos, para posteriormente hacer la conexión final con elasticsearch y registrar de la manera que se solicita en el "mapping".

Comando para ejecutar dicho archivo de configuración es:

```
/usr/share/logstash/bin/logstash -f /home/aibg/Descargas/ultimo/proy.conf --path.settings /etc/logstash
```

2.1 Proceso del fichero CSV

Se empieza con un fichero que en su input recoja solo un archivo csv, ignorando el resto de csv's del directorio de Datos. Con este método trabajar será más sencillo, en vez de esperar la carga de todos los productos y archivos. Después de leer este archivo gracias a la función del "input", se imprimen los datos por consola para controlar que salida de datos se obtiene. Y con esta visión ir modificando el algoritmo en el "filter" hasta lograr el formato deseado.

2.1.1 Filtrado

Ya que todo el mensaje se aglomera en un "message", siempre en nuestro csv se van a marcar 7 campos, cada uno corresponde a una posición. También se añade el delimitador que en este caso es ";". Así que utilizando el atributo "CSV" se le indica la propiedad "separador" y su delimitador. La propiedad "columns" se le indica en una array que ha cada posición le añada un nombre a cada uno de estos campos. Al ejecutar logstash se puede identificar los datos definidos en el formato "clave-valor".

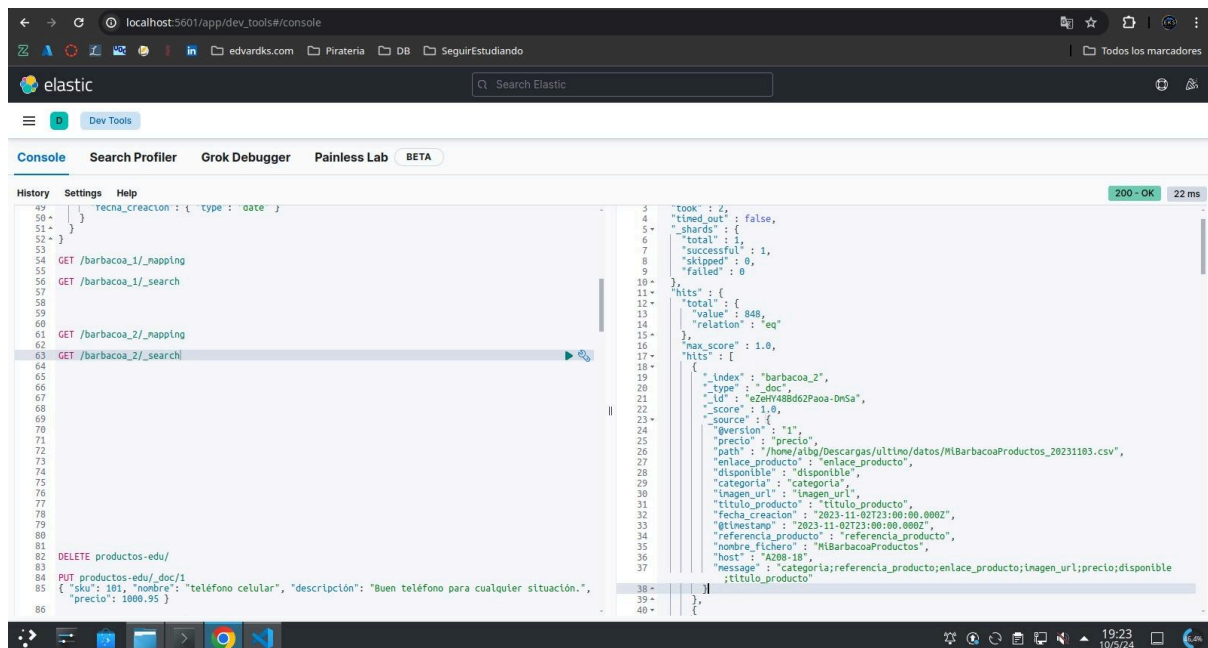
Se observa que hay campos que son irrelevantes como "host", "@version"... y por otro lado el "@timestamp", pone la fecha actual.

Para solucionar se siguen estos pasos:

- Inferir en "@timestamp" la fecha de creación de cada CSV: Puesto que no existe ninguna columna con la fecha de creación, si se encuentra en el atributo "path", que posteriormente se elimina. La solución seguida es usando el atributo "grok" y la propiedad "match" indicarle que de "path" coja el nombre del fichero, y la fecha, que está en formato "yyyMMdd". Este luego es inferido en el "@timestamp" con el atributo "mutate" y la propiedad "replace".
- Eliminar Datos innecesarios: Utilizado el atributo "mutate" y la propiedad "remove_fields" se indica en una array las columnas a eliminar.

Al final, después de comprobar que por consola se imprime el dato cómo se desea, del único fichero mencionado en el input, se procede a cambiar el output para que se registre en elasticsearch. Simplemente se modifica en el “*output*” el atributo “*elasticsearch*” y la propiedad “*host*” dónde irá la dirección de nuestra máquina e “*index*” dónde se indica el índice (que ha de crear o encontrar para hacer el “*ingest*”).

Se hace una prueba sin crear el “*mapping*” para comprobar que columnas son creadas y que columnas no. Cómo hace automáticamente la colocación de los atributos por “*clave-valor*”. Se observa un resultado adecuado y esperado.



En la imagen anterior se observa que los datos que nos mostraba al hacer el output en consola, de igual manera se registran aquí. La diferencia es que se le añade el “*type*” del documento en “*_doc*”, un “*id*”, el “*index*” al que pertenece, y un “*score*”.

En los datos de “*source*” se colocan estos. Pero en esta parte todavía en filter no se han añadido la eliminación de los campos innecesarios (@version, path, fecha_creacion que viene duplicada con el timestamp, host). Posteriormente se eliminará.

```
input { file {
  path => "/home/aibg/Descargas/ultimo/datos/*.csv"
  start_position => "beginning"
  sincedb_path => "/dev/null" }}

filter {
  csv {
    separator => ","
    columns => ["categoria", "referencia_producto", "enlace_producto", "imagen_url", "precio", "disponible", "titulo_producto"]
  }
  grok {
    match => { "path" => "/(?<nombre_fichero>[^\s/]+)_?(?<fecha_creacion>[d+])\.csv" }
  }
  date {
    match => [ "fecha_creacion", "yyyyMMdd" ]
    target => "@timestamp"
  }
  mutate {
    replace => { "fecha_creacion" => "%{@timestamp}" }
  }
  mutate {
    remove_field => [ "@version", "fecha_creacion", "path", "host" ]
  }
}

output { elasticsearch {
  hosts => ["localhost:9200"]
  index => "barbacoa_6" }}
```

Este es el archivo proy.conf final. Será el file que se ejecutará para hacer la carga de datos (“*ingest*”) al servicio de elasticsearch.

2.2 Proceso del fichero JSON

Recuerda que ya se reciben los datos desde un consumidor de kafka que se encuentra en el servidor. Ahora estos datos se han escrito en un JSON, y de manera automatizada, crea un contenedor Docker el cual tendrá un logstash efímero, que ejecuta el archivo de configuración que se indica.

Comando a ejecutar:

```
comando_docker = "
docker run --rm -it --network elog \
-v ./logstash.yml:/usr/share/logstash/config/logstash.yml \
-v ./pipelines.yml:/usr/share/logstash/config/pipelines.yml \
-v ./nuevo_resultado.json:/home/resultado.json \
-v ./proy.conf:/usr/share/logstash/config/proy.conf \
docker.elastic.co/logstash/logstash:7.17.21 \
-f /usr/share/logstash/config/proy.conf
"
```

Esto creará un contenedor efímero, conectado a la red interna "mi_red" dónde se encuentra elasticserach. Copiará en el contenedor los ficheros indicados en las líneas -v (el json con los datos recogidos, y el archivo de configuración de logstash) en las carpetas indicadas, por último indica al logstash que ejecute dicho archivo de configuración.

```

14
15 json_output = json.dumps(product_list, indent=4, ensure_ascii=False)
16
17 with open("nuevo_resultado.json", "w", encoding="utf-8") as file:
18     file.write(json_output)
19
20 print("JSON guardado en 'productos.json'")
21
22 comando = """docker run --rm -it --network elog \
23 -v ./logstash.yml:/usr/share/logstash/config/logstash.yml \
24 -v ./pipelines.yml:/usr/share/logstash/config/pipelines.yml \
25 -v ./nuevo_resultado.json:/home/resultado.json \
26 -v ./proy.conf:/usr/share/logstash/config/proy.conf \
27 docker.elastic.co/logstash/logstash:7.17.21 \
28 -f /usr/share/logstash/config/proy.conf
29 """
30
31

```

```

[0] "jsonparsefailure"
},
"@version" => "1",
"message" => "\n{'titulo_producto': 'Bancada Menorca Granito', 'categoria': 'Barbacoas de Obra', 'referencia_producto': '150132', 'enlace_producto': 'https://mibarba
coa.com/es/barbacoas-de-obra/bancada-menorca-white#/475-color-blanco', 'imagen_url': 'https://mibarbacoa.com/6835-home_default/bancada-menorca-white.jpg', 'precio': 305.0, 'dis
ponible': True}\n",
"@timestamp" => 2024-05-17T14:41:53.547Z,
"path" => "/home/resultado.json",
"host" => "b6b7f4faf4f8"
}
{
  "tags" => [
    [0] "jsonparsefailure"
  ],
"@version" => "1",
"message" => "\n{'titulo_producto': 'Módulo Bancada Lloret', 'categoria': 'Barbacoas de Obra', 'referencia_producto': '150002M B', 'enlace_producto': 'https://mibar
bacoa.com/es/barbacoas-de-obra/modulo-bancada-lloret#/747-color_ladrillo-ladrillo_blanco', 'imagen_url': 'https://mibarbacoa.com/4934-home_default/modulo-bancada-lloret.jpg', '
precio': 356.0, 'disponible': True}\n",
"@timestamp" => 2024-05-17T14:41:53.580Z,
"path" => "/home/resultado.json",
"host" => "b6b7f4faf4f8"
}
}

```

Aquí se observa que se ha ejecutado el contenedor de manera adecuada e imprime los datos del json creado por el servidor, y que posteriormente el tratado de datos para luego pasarlo a logstash.

Ahora toca tratar los datos con “filter” y hacer la inyección en elasticsearch.

```

12 filter {
13   grok {
14     match => [ "fecha_creacion", "yyyyMMdd" ]
15     target => "@timestamp"
16   }
17   date {
18     match => [ "fecha_creacion", "yyyyMMdd" ]
19     target => "@timestamp"
20   }
21   mutate {
22     replace => { "fecha_creacion" => "%{@timestamp}" }
23   }
24   mutate {
25     remove_field => ["@version", "host", "tags"]
26   }
27 }
28
29 output {
30   elasticsearch {
31     hosts => ["localhost:9200"]
32     index => "barbacoa_7"
33   }
34 }

```

Podemos comprobar cómo se obtienen las columnas de manera correcta. Con las ediciones en el “filter” de los datos, y la inyección al contenedor de elasticsearch.

El código del archivo de configuración es el siguiente:

```

input {
  file {
    path => "/home/resultado.json"
    start_position => "beginning"
    sincedb_path => "/dev/null"
    codec => "json"
  }
}

filter {
  grok {
    match => { "message" => "\{'titulo_producto': '%{DATA:titulo_producto}', 'categoria': '%{DATA:categoria}', 'referencia_producto': '%{DATA:referencia_producto}', 'enlace_producto': '%{URI:enlace_producto}', 'imagen_url': '%{URI:imagen_url}', 'precio': %{NUMBER:precio}, 'disponible': %{WORD:disponible}\}" }
  }
  date {
    match => [ "@timestamp", "yyyyMMdd" ]
    target => "@timestamp"
  }
  mutate {
    replace => { "@timestamp" => "%{@timestamp}" }
  }
  mutate {
    remove_field => ["@version", "host", "tags"]
  }
}

output {
  elasticsearch {
    hosts => ["localhost:9200"]
    index => "barbacoa_7"
  }
}

```

```
}}}
```

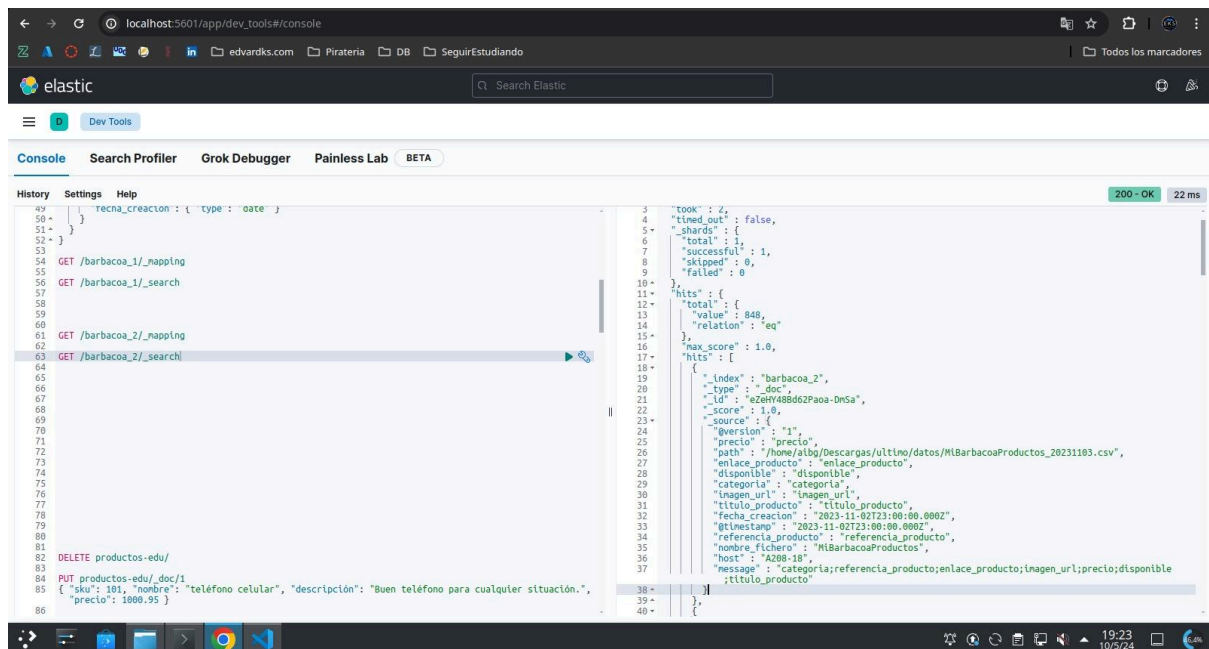
Podemos observar que cambia el grok, pero el resto de atributos es el mismo.

3. Creación de Mapping

3.1 Procedimiento y toma de decisión

La estructuración del “*mapping*” es personalizado adaptado a los datos del topping. El objetivo de hacer esto es marcar el tipo de los datos.

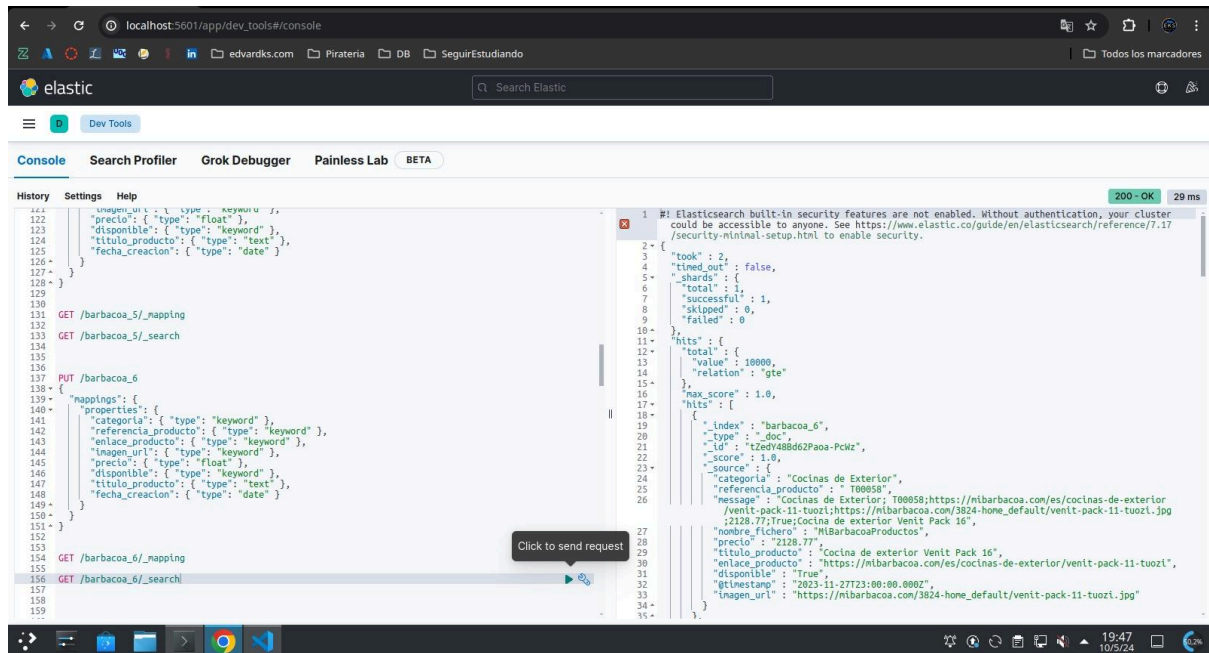
Es cierto que elasticsearch lo puede hacer de forma automática. Se hace la prueba de ambas maneras de inyección de datos. Pero las buenas prácticas aconsejan seguir la metodología del “*mapping*”.



En esta imagen automáticamente ya se crean los campos.

En el siguiente trozo del código se observa la decisión final, tomada para el tipo de datos que se espera recoger.

```
PUT /barbacoa_6
{
  "mappings": {
    "properties": {
      "categoria": { "type": "keyword" },
      "referencia_producto": { "type": "keyword" },
      "enlace_producto": { "type": "keyword" },
      "imagen_url": { "type": "keyword" },
      "precio": { "type": "float" },
      "disponible": { "type": "keyword" },
      "titulo_producto": { "type": "text" },
      "fecha_creacion": { "type": "date" }
    }
  }
}
```



Después de ejecutar el archivo `proy.conf`, observamos los datos (y ya aplicado el `"remove_fields"` de los campos innecesarios) y se van a ajustar a los tipos que le hemos indicado en el mapping.

```
/usr/share/logstash/bin/logstash -f /home/aibg/Descargas/ultimo/proy.conf
nf --path.settings /etc/logstash
```

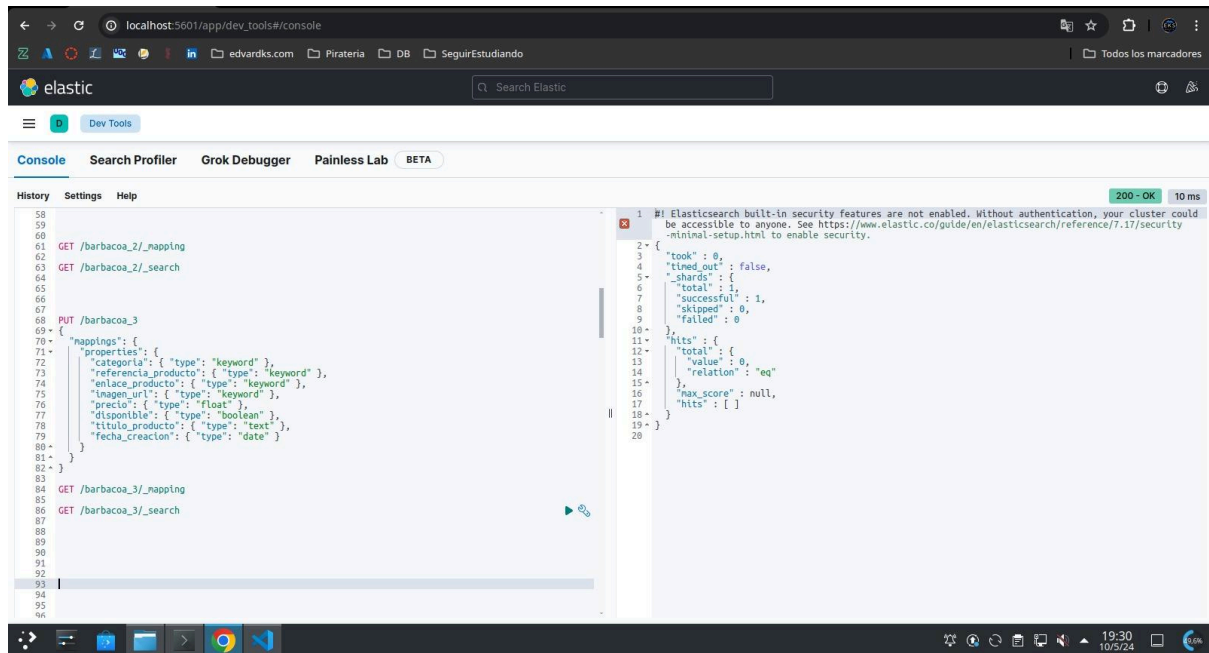
Este comando ejecuta el archivo de `.conf` para que acceda a los CSV, en este caso, y se indica dónde está el servicio de logstash.

3.2 Contratiempo

El primer contratiempo fue al cargar los datos. Se generaba un error el `"ingest"`, ya que la columna `"disponibilidad"` no recogía los datos, indicando que el tipo de datos no corresponde al esperado (especificado en el `"mapping"`).

Sucede que al ser recogido los datos con python, este guarda el `"True"` y `"False"` capitalizados, pero elasticsearch entiende los booleanos con minúsculas (`"true"`, `"false"`). Varias formas de resolver esto serían las siguientes:

- Al hacer web Scraping guardar con 0,1 o true y false
- Al hacer el filter hacer un mutate para modificar los datos de manera que lo entienda elasticsearch
- Al hacer el mapping cambiar el tipo de dato de `"disponibilidad"` a `"keyword"`



En la imagen podemos observar que después de hacer el GET .../_search nos trae los datos como null, y en consola podemos observar que el mensaje de error nos indica *“type of disponibilidad is wrong”*.

```
[2024-05-14T17:07:21,099][WARN][logstash.outputs.elasticsearch][main][4f2c5ce33738a886054f6b88b758fcec848056ff8ca94e82fa186e73a388e702] Could not index event to Elasticsearch. {:status=>400, :action=>["index", {:id=>nil, :index=>"barbacoa_3", :routing=>nil}], {:referencia_producto=>"ESTELA12", :titulo_producto=>"Estufa de Pellets Estela", :nombre_fichero=>"MiBarbacoaProductos", :imagen_url=>"https://mibarbacoa.com/4817-home_default/estufa-de-pellets-estela.jpg", :categoria=>"Estufas", :timestamp=>"2023-11-02T23:00:00.000Z", :enlace_producto=>"https://mibarbacoa.com/es/estufas/estufa-de-pellets-estela#/453-color-marron/744-ferlux_potencia-12_kw", :disponible=>"True", :precio=>"1883.97", :message=>"Estufas; ESTELA12;https://mibarbacoa.com/es/estufas/estufa-de-pellets-estela#/453-color-marron/744-ferlux_potencia-12_kw;https://mibarbacoa.com/4817-home_default/estufa-de-pellets-estela.jpg;1883.97;True;Estufa de Pellets Estela"}], :response=>{"index"=>{"_index"=>"barbacoa_3", "type"=>"doc", "_id"=>"Azykd48Bd62Pa0a-0Ftv", "status"=>400, "error"=>{"type"=>"mapper_parsing_exception", "reason"=>"failed to parse field [disponible] of type [boolean] in document with id 'Azykd48Bd62Pa0a-0Ftv'. Preview of field's value: 'True', :caused_by"=>{"type"=>"illegal_argument_exception", "reason"=>"Failed to parse value [True] as only [true] or [false] are allowed."}}}}}
```

No se aprecia el error, así que también lo voy a insertar a continuación:

```
[2024-05-14T17:07:21,099]
[WARN][logstash.outputs.elasticsearch][main][4f2c5ce33738a886054f6b88b758fcec848056ff8ca94e82fa186e73a388e702] Could not index event to Elasticsearch. {:status=>400, :action=>["index", {:id=>nil, :index=>"barbacoa_3", :routing=>nil}],
{"referencia_producto"=>" ESTELA12", "titulo_producto"=>"Estufa de Pellets Estela",
"nombre fichero"=>"MiBarbacoaProductos", "imagen_url"=>"https://mibarbacoa.com/4817-home_default/estufa-de-pellets-estela.jpg",
"categoria"=>"Estufas", "@timestamp"=>"2023-11-02T23:00:00.000Z",
"enlace_producto"=>"https://mibarbacoa.com/es/estufas/estufa-de-pellets-estela#/453-color-marron/744-ferlux_potencia-12_kw",
"disponible"=>"True", "precio"=>"1883.97", "message"=>"Estufas; ESTELA12;https://mibarbacoa.com/es/estufas/estufa-de-pellets-estela#/453-color-marron/744-ferlux_potencia-12_kw;https://mibarbacoa.com/4817-home_default/estufa-de-pellets-estela.jpg;1883.97;True;Estufa de Pellets Estela"}],
:response=>{"index"=>{"_index"=>"barbacoa_3", "type"=>"doc", "_id"=>"Azykd48Bd62Pa0a-0Ftv", "status"=>400,
"error"=>{"type"=>"mapper_parsing_exception", "reason"=>"failed to parse field [disponible] of type [boolean] in document with id 'Azykd48Bd62Pa0a-0Ftv'. Preview of field's value: 'True',
"caused_by"=>{"type"=>"illegal_argument_exception", "reason"=>"Failed to parse value [True] as only [true] or [false] are allowed."}}}}}
```

Se observa una alerta de “warning”, a continuación la línea que se ha intentado insertar, y seguidamente el motivo del mensaje, *“failed to parse field [disponible] of type [boolean]”*

4. Creación del “Index-Pattern”

Para poder acceder a los datos registrados en el índice, es necesaria la creación de este concepto, sirve para facilitar la visualización y el análisis de datos en elasticsearch.

Este index-pattern define cómo se agrupan y visualizan los datos, mejorando la eficiencia de las consultas y permitiendo personalizar las visualizaciones.

Para crearlo, hay que acceder a "*Stack Management*" > "*Index Patterns*". En "*Kibana*" se selecciona el índice, se define el patrón, se eligen los campos relevantes y se guarda la configuración.

Este proceso es esencial para explorar y analizar datos de manera efectiva en "*Kibana*".

Create index pattern

Name:

Use an asterisk (*) to match multiple characters. Spaces and the characters , / ? * < > | are not allowed.

Timestamp field:

Select a timestamp field for use with the global time filter.

[Show advanced settings](#)

✓ Your index pattern matches 1 source.

barbacoa_4 Index

Rows per page: 10

En esta captura se puede comprobar un ejemplo al crear un index pattern, y es importante relacionarlo con un index existente. Y también en timestamp para indicarle su formato de fecha, indicaremos nuestra columna de las fechas.

Management

barbacoa_6*

Time field: @timestamp

View and edit fields in **barbacoa_6***. Field attributes, such as type and searchability, are based on [field mappings](#) in Elasticsearch.

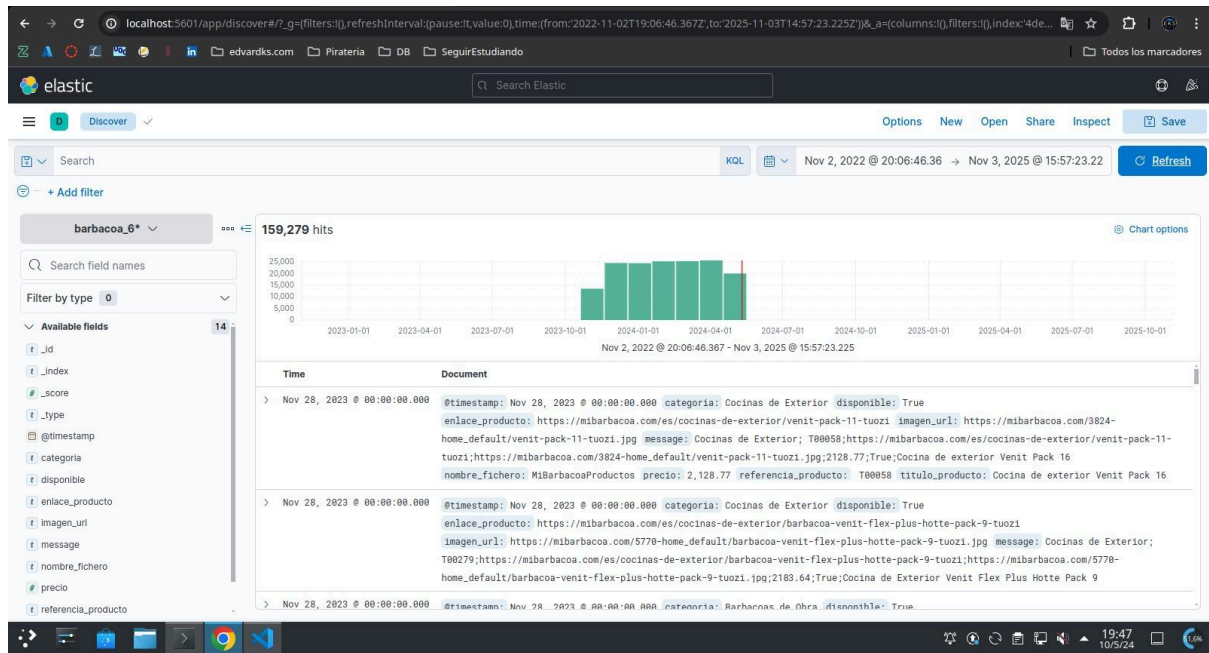
Fields (18) Scripted fields (0) Field filters (0)

Search

All field types Add field

Name	Type	Format	Searchable	Aggregatable	Excluded
@timestamp	date		•	•	
_id	_id		•	•	
_index	_index		•	•	
_score					
_source	_source				
_type	_type		•	•	
categoria	keyword		•	•	
disponible	keyword		•	•	

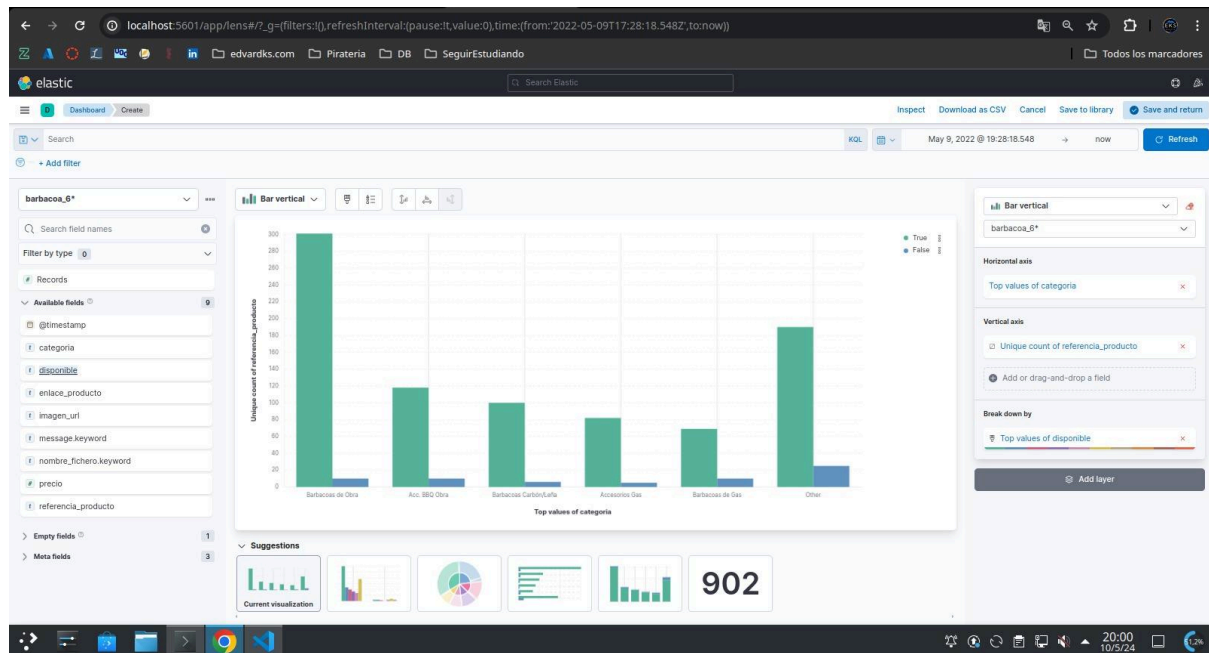
En la imagen como ha sido creado el index-pattern relacionado con el índice dónde hemos hecho la ingesta de datos y las columnas que contiene.



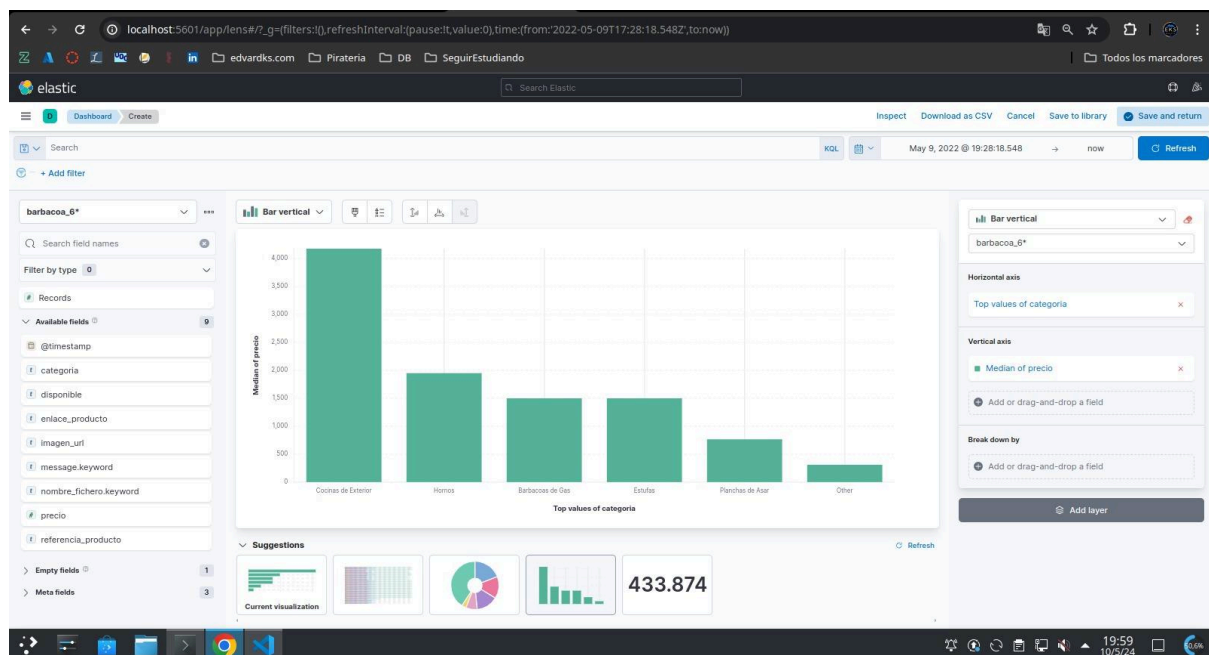
En la imagen anterior se puede observar la carga de datos en el apartado “Discover”, y gracias a la fecha incluida de @timestamp, que es la fecha de creación de cada archivo, recordamos que en el “filter” asignamos a cada producto su fecha en el que se ha buscado, siendo asignada a cada producto en su “json”.

Debajo de la gráfica se encuentran algunos ejemplos de productos que se han hecho desde la ETL (Extracción, transformación y carga) al index.

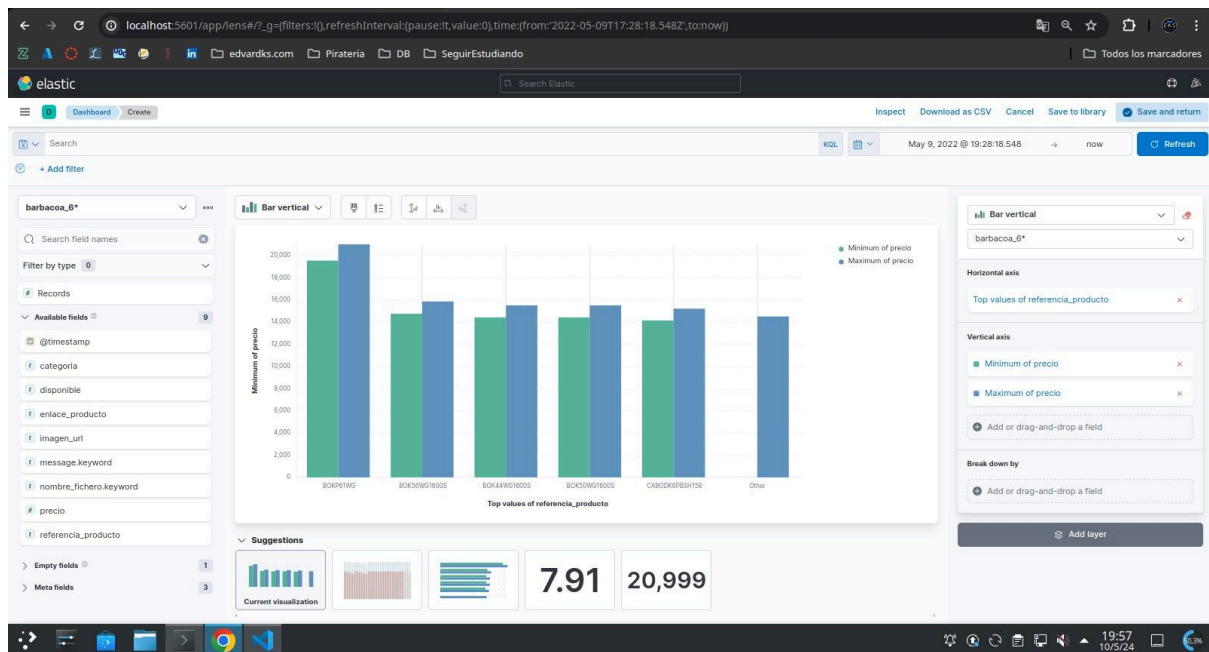
5. Dashboard



En este primer dashboard analizamos las categorías de los productos, la cantidad de productos que hay de cada categoría, en las barras verdes veremos productos disponibles y las azules productos de su misma categoría pero no disponibles.



Seguimos analizando las categorías, pero en este caso vamos a buscar la categoría con los productos más caros. Por ello colocamos en el axis "x" las categorías y en la "y" la media de los precios. Como conclusión obtenemos que los productos más caros suelen ser los de la categoría "cocinas de exterior", y de la categoría "planchas para asador" la más barata.

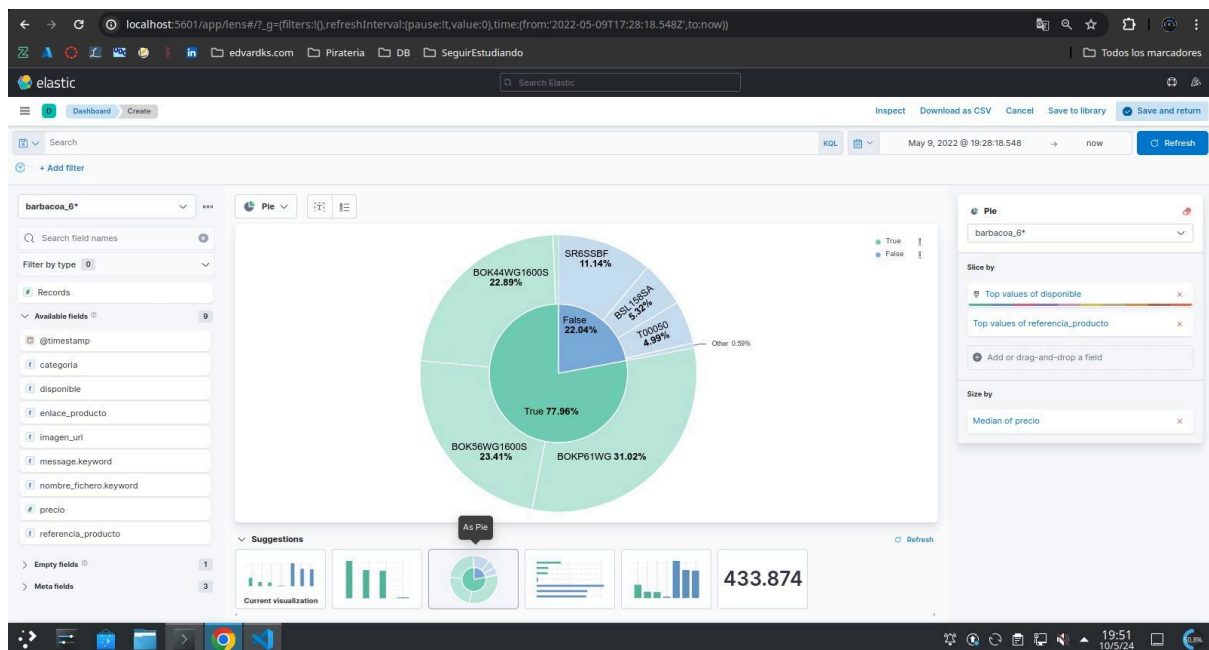


Ahora localizamos en el eje "x" los 5 artículos más caros, y veremos su valor más caro contra su valor más bajo. Según el dashboard tenemos una varianza de un mismo producto de un 7.91% de media y por ejemplo vemos una tendencia de varianza en los precios del primer artículo de un 7.14% aproximadamente.

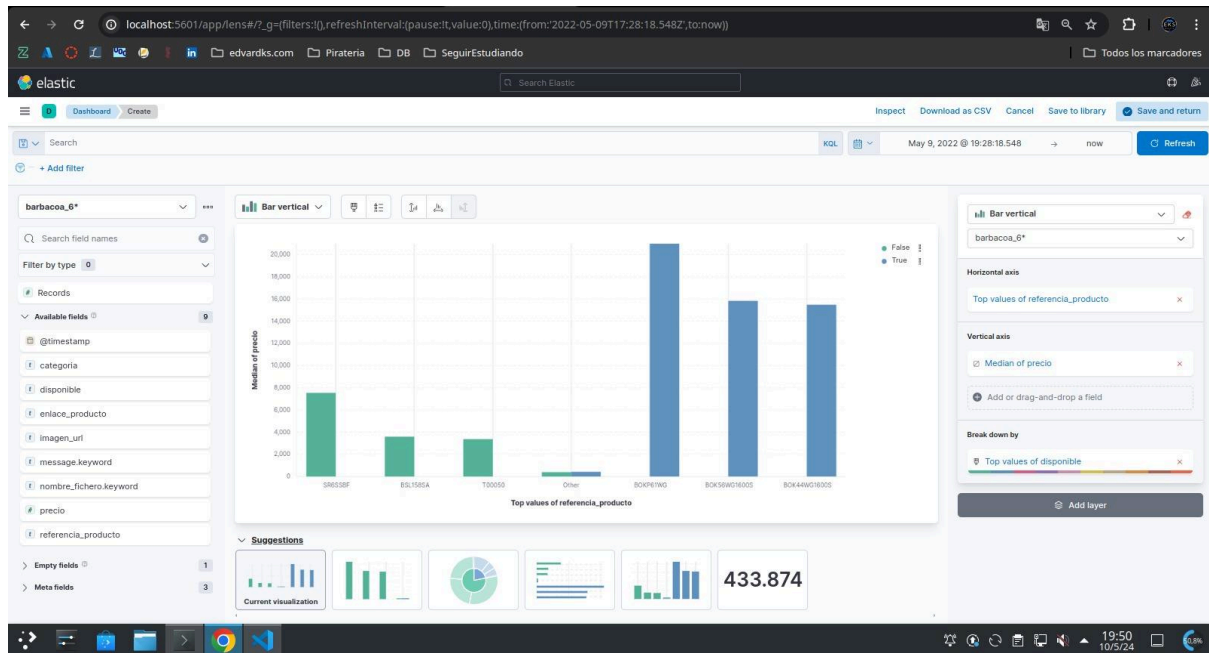
($\text{porcentaje} = \frac{\text{ValorNuevo} - \text{ValorOriginal}}{\text{ValorOriginal}}$)

Esto será debido a que suele entrar algún descuento y oferta.

Más adelante analizaremos en qué fechas suele suceder este cambio de precios.

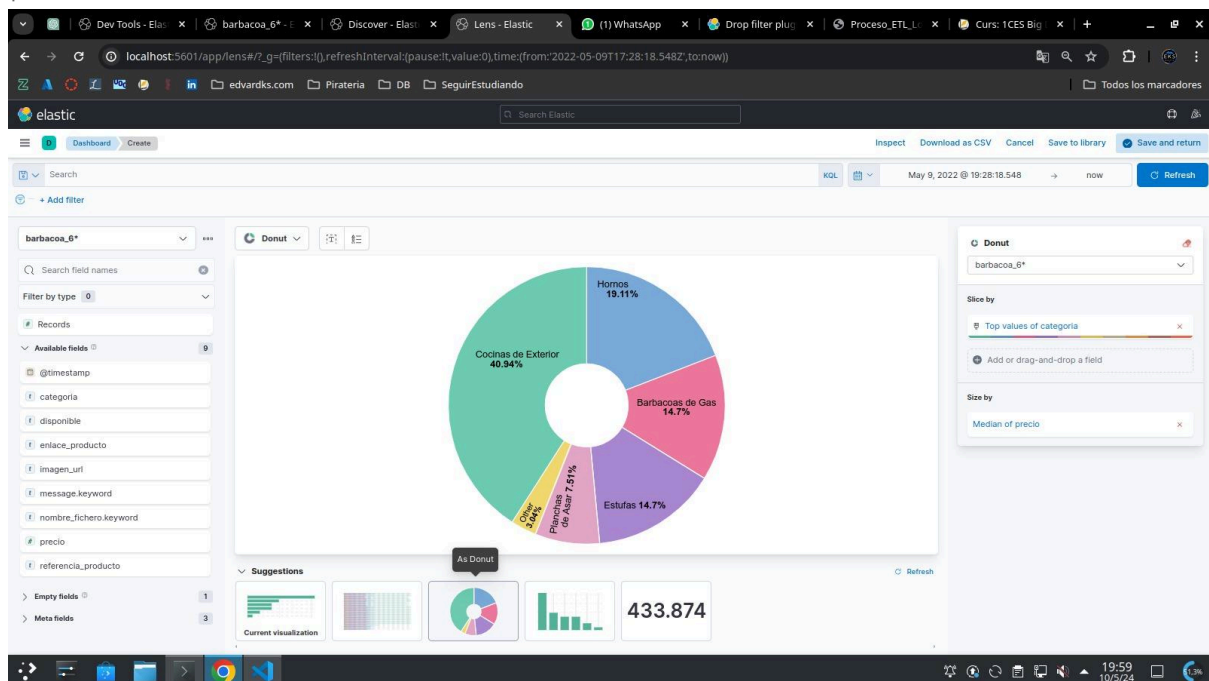


Para confirmar los productos más veces disponibles frente a los productos que más veces no están disponibles, sacamos el top 3 de cada una de ellas, e incluso vemos el porcentaje de disponibilidad de cada uno de ellos. En conclusión podemos decir que está web suele mantener sus productos accesibles y actualizados.



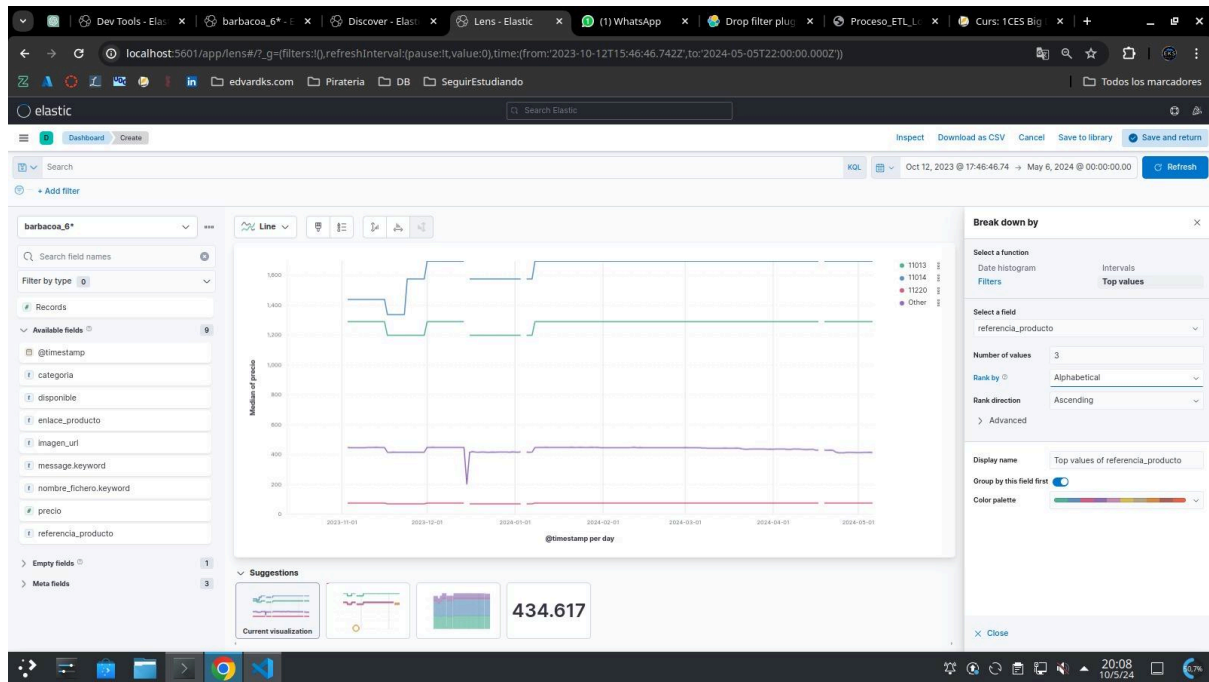
Aquí sacando la media de los precios de todos los productos, vemos que de normal los productos más caros son los que más tiempo suelen estar disponibles, y los productos de un precio más asequible no.

La conclusión puede ser que la web vende mucho producto que se encuentra en ese rango de la media, y se suele quedar sin stock.



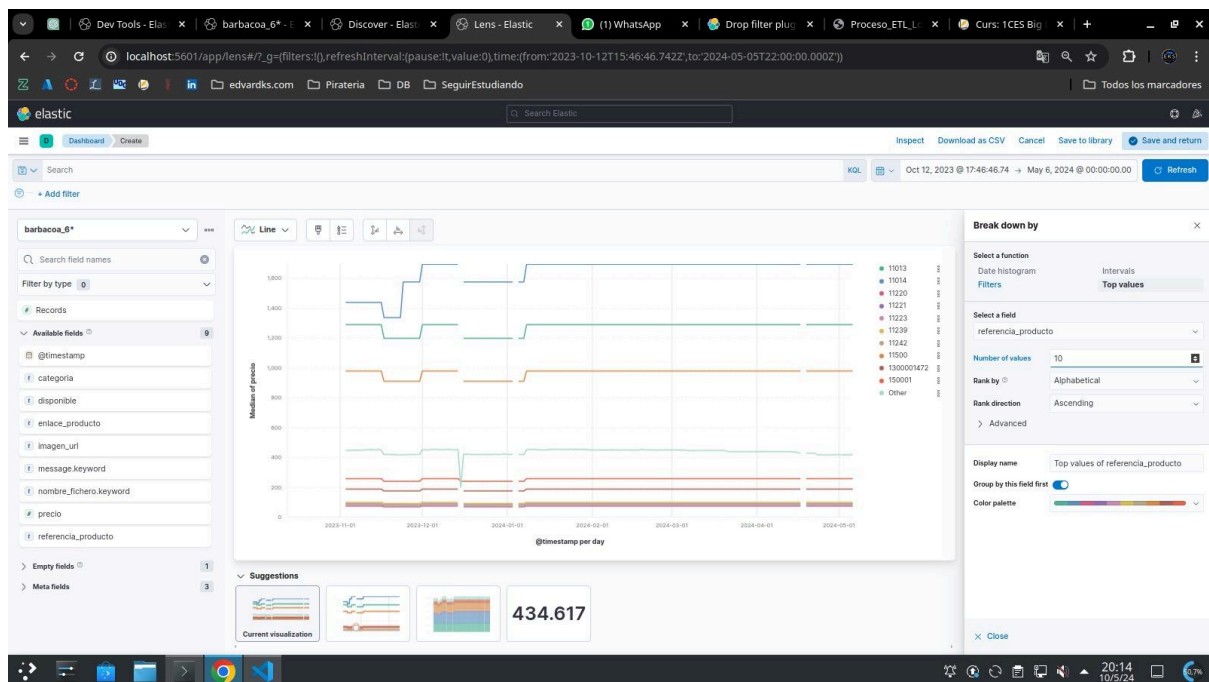
En esta gráfica de "Donut" vemos todas las categorías, y que en el stock de la tienda, predomina la categoría de "Cocinas de Exterior".

La conclusión tiene un peso importante, y es que tratamos con una web que se dedica a las barbacoas, estas suelen situarse en el exterior de todas las casas de campo o chalets.



Sacamos las gráficas del top 3 (orden alfabéticamente) de productos en el periodo de tiempo que tenemos los datos, pudiendo observar que a principios de Diciembre suelen subir, y a mediados de este, los precios toman una bajada.

En conclusión, está en el mercado, cerca de fechas importantes, los precios pueden pegar una subida, y justo en los días clave, ofertar a la gente el mismo producto pero con una llamada a que "ha habido una oferta".



Aquí sacando la media de los precios de todos los productos, vemos que de normal los productos más caros son los que más tiempo suelen estar disponibles, y los productos de un precio más asequible no.

La conclusión puede ser que la web vende mucho producto que se encuentra en ese rango de la media, y se suele quedar sin stock.

Conclusión personal

Quiero mencionar que para lograr todo esto sin la mínima experiencia, es prácticamente muy complejo, sin mencionar la dedicación de todo el tiempo que requiere. Pero no solo le he dedicado el tiempo que se merece este proyecto, porque he disfrutado en el desarrollo, sino que quiero destacar la ayuda del profesor encargado de la asignatura.

En mi caso he abordado las dos formas que se pueden consumir los datos, una forma más sencilla que sería "[1.1 Recogida de los datos](#)" y al terminarlo con un gran éxito esta parte, ya se daría por concluido el proyecto. Pero he querido aplicarse más allá de lo solicitado y dedicar el tiempo a lo que sería la idea principal del proyecto, denominada "[1.2 Recogida via consumidor Kafka](#)" y que culminaría todo lo aprendido en este curso de especialización en "big data".

Dejando a un lado los contratiempos que he sufrido por falta de experiencia, el mayor reto ha sido la optimización y automatización de todo el proceso de la ETL, el cual queda resumido en una simple línea de comando. Estoy muy orgulloso de haber hecho una automatización real, pudiendo demostrar el ahorro de tiempo con la ayuda de este, y la sencillez para manos más inexpertas que lo necesiten.

Tengo presente que todavía se puede mejorar mucho más, pasándolo a un archivo Dockerfile, que será un reto personal y profesional que en un futuro me encantaría abordar.

Ha sido un proyecto muy enriquecedor, desde sus comienzos en octubre de 2023, hasta el final del curso y el desarrollo de esta ETL.

Webgrafía

Elastic.co:

- Construcción de los contenedores:
 - <https://www.elastic.co/guide/en/logstash/7.17/docker.html>
 - <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/docker.html>
- Configuración de los pipelines de Elasticsearch:
 - <https://www.elastic.co/guide/en/logstash/current/multiple-pipelines.html>
- Plugins de input, filter y output:
 - <https://www.elastic.co/guide/en/logstash/current/plugins-filters-age.html>
 - <https://www.elastic.co/guide/en/logstash/current/plugins-outputs-boundary.html>
 - https://www.elastic.co/guide/en/logstash/current/plugins-inputs-azure_event_hubs.html
 -

Resolución de errores sencillos:

- <https://chatgpt.com/>
- <https://www.phind.com/search?home=true>

Resolución de errores en desarrollo y experiencia de otros usuarios:

- <https://stackoverflow.com/questions/78487898/logstash-parsing-error-but-found-a-concrete-value>
- <https://stackoverflow.com/questions/24606059/does-logstash-has-a-limit-size-for-each-event-message>
- <https://stackoverflow.com/>
- <https://discuss.elastic.co/t/logstash-hangs-with-error-sized-queue-timeout/59031>
- <https://senoritadeveloper.medium.com/create-logstash-config-to-feed-elasticsearch-from-kafka-6311f536525a>
- <https://discuss.elastic.co/t/logstash-reading-rate-is-max-8-9k-per-second/127977>

Documentación IES Abastos por “Vicent Tortosa”:

- Doc del proyecto: <https://aules.edu.gva.es/fp/mod/resource/view.php?id=5853800>
- Mapping: <https://aules.edu.gva.es/fp/mod/resource/view.php?id=5831208>
- Caso de Uso: <https://aules.edu.gva.es/fp/mod/resource/view.php?id=5830950>
- Instalación y Funciones: <https://aules.edu.gva.es/fp/mod/resource/view.php?id=5815670>
- Logstash 2024: <https://aules.edu.gva.es/fp/mod/resource/view.php?id=5815664>