IPCA

Escola Superior Tecnologia

Integração Sistemas de Informação

Joel Phillippe Melo Figueiras, nº20809

Trabalho sobre a orientação de:

Professor Óscar Ribeiro

2022/2023

Indíce de figuras

Indíce

1 Introdução

1.1 Contextualização

Este relatório foi realizado no âmbito do trabalho prático da unidade curricular de Integração de Sistemas de Informação, inserida no plano de estudos do curso de engenharia de sistemas informáticos, lecionada pelo docente Óscar Ribeiro, tendo como principal objetivo documentar e auxiliar todo o trabalho realizado em volta deste trabalho. Este trabalho prático consiste na aplicação do conceito de processos ETL, que tem como objetivo extrair dados de uma origem e carregar os dados transformados num determinado dataset destino, o mosquitto que é um intermediário de mensagens de código aberto que implementa as versões 5.0, 3.1.1 e 3.1 do protocolo MQTT e por fim o Node Red que é uma plataforma baseada em fluxos, foi desenvolvida para interagir com dispositivos de hardware, API's e serviços online com vista a simplificar os sistemas IoT (Internet of Things).

1.2 Motivação e Objetivos

A motivação na realização deste trabalho é o desafio que nos é proposto sendo que podemos aumentar os nossos conhecimentos e investigar ainda mais do que aquilo que foi aprendido durante as aulas sobre o processo ETL, mosquitto e o Node Red, sendo ferramentas muito utilizadas na atualidade.

1.3 Objetivo

O objetivo deste trabalho consiste na implementação do processo ETL para a extração dos dados, o envio dos resultados obtidos através do broker Mosquitto e a apresentação dos resultados obtidos em dashboard desenvolvidos na ferramenta Node Red. Foi explorado um caso de uso sobre o comércio de automóveis, que tem como objetivo apresentar ao comerciante algumas dicas ou estratégias de negócio resultantes a partir dos dados obtidos referentes á sua empresa e uma visão mais apelativa e real dos dados.

2 Csharp

2.1 Leitura e análise dos dados

O ponto inicial do projeto passa por um programa desenvolvido no Visual Studio Code em linguagem Csharp, para efetuar a limpeza e verificação dos dados que tem como input um ficheiro csv designado por "car_details.csv".
Os dados disponibilizados referem-se às seguintes colunas:

- Nome
- Ano
- Preço
- Km
- Combustível
- tipo
- Transmissão
- Dono

Para controlar e remover dados fora do contexto recorreu-se a uma expressão regular como mostra a figura seguinte.

```
string er = @"[^,]+,([1]|[2])[0-9]{3},[0-9]*,[0-9]*,[A-Za-z,]+,[^\n]+";
```

Figura 1: Expressão Regular

Conseguido então a expressão regular para o controlo dos dados é feita a leitura do ficheiro para a verificação e exclusão dos mesmos.

```
foreach (string line in File.ReadLines(ficheiro))
{
    csvString = line;
    Regex g = new Regex(er);
    Match m = g.Match(csvString);

    if (m.Success)
    {

        string[] campos = m.Value.Split(',');
        string nome = campos[0];
        int ano = Convert.ToInt32(campos[1]);
        int preco = Convert.ToInt32(campos[2]);
        int km = Convert.ToInt32(campos[3]);
        string fuel = campos[4];
        string transmissao = campos[6];
        string n_dono = campos[7];
        string tipo_dono = campos[5];
        Veiculo veiculo = new Veiculo(nome, ano, preco, km, fuel, transmissao, n_dono, tipo_dono);
        listaVeiculos.Add(veiculo);

    }
    else
    {
        Console.WriteLine($"{line}");
    }


}
```

Figura 2: Leitura, separação e armazenamento em memória do ficheiro "car_details.csv"

Como apresenta a figura 2, recorri ao método "ReadLines" que pertence á class "FILE" onde existem inumeros métodos para podermos trabalhar com ficheiros. Enquanto a leitura, para cada linha do ficheiro é instanciada a classe "Regex" para inicializar a expressão regular da figura 1 e a class "Match" onde é feita a comparação da linha do ficheiro com a expressão regular, no caso de combinarem, é feita a separação do conteúdo com o "Split" e armazenado numa lista referente á estrutura de dados "Veiculo", como mostra a figura 27.

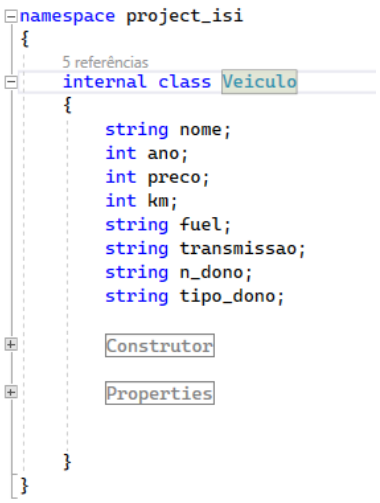

Figura 3: Estrutura dados de um Veiculo

2.2 Escrita dos dados em ficheiro json

Ao fim da verificação e armazenamento em memória dos dados referentes a um veículo, é então criado um ficheiro json para ser interpretado na ferramenta mais á frente que será entretando abordada.

```
jsonString = System.Text.Json.JsonSerializer.Serialize(listaVeiculos);
jsonString = "{\"Carro\": " + jsonString + "}";


if (!File.Exists(destinojson))
{
    File.WriteAllText(destinojson, jsonString);

}
```

Figura 4: Conversão da lista Veiculos num ficheiro Json

Para criar um ficheiro json é então feito uma Serialização da lista de veículos, convertendo a lista de veículos num json em formato de string. Logo de seguida é então criado um ficheiro json é escrito no mesmo a string que contém os dados em formato json.

2.3 Escrita dos dados em ficheiro xml

Por fim quanto ao programa em C# o ficheiro Json é convertido num ficheiro xml. Para isso é criado um nó da class XNode onde é feito a deserialização do ficheiro Json com a indicação do seu elemento raiz que neste caso se chama de "Carros" como mostra a figura

```
if(!File.Exists(destinoxml))
{
    XNode node = JsonConvert.DeserializeXNode(jsonString, "Carros");
    File.WriteAllText(destinoxml, node.ToString());
}
```

Figura 5: Deserialização do ficheiro Json, criação e escrita do ficheiro xml

3 Pentaho kettle

O Pentaho Data Integration, também conhecido por Kettle é uma ferramenta de código aberto desenvolvida em Java, para a extração, transformação e carga de dados. Vejamos de seguida todas as extrações, transformações e carregamento de dados referente ao Json de Output do Visual Studio.

3.1 Armazenamento dos dados em base dados MongoDB

A escolha do servidor para a base dados foi o MongoDB, devido á necessidade de guardar em várias coleções um grande numero de dados em formato Json de forma não relacional. Então o primeiro passo ETL a ser feito é o input do ficheiro Json resultante do output do Visual Studio, fazer uma filtragem dos veículos por quilómetros e criar a base dados designada por "Carros", com duas coleções diferentes, uma com veículos superiores a 100000km e outra com os veículos inferiores a 100000km.

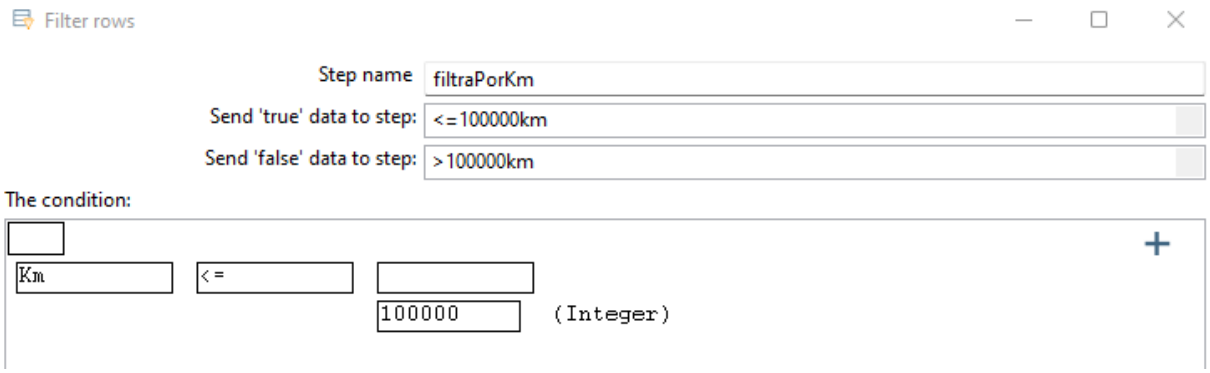

Figura 6: Filtragem dos veículos por quilómetros

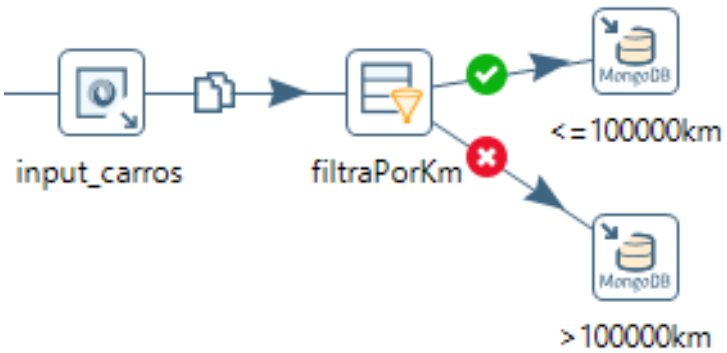

Figura 7: Filtragem e Armazenamento dos veículos em base dados MongoDB

3.2 Importação dos dados em base dados MongoDB e exportação para ficheiro xml

Na seguinte transformação é realizado um input de dados de todos os veículos armazenados em base dados, são feitos dois pedidos de entrada, porque os veículos foram anteriormente armazenados em coleções de dados diferentes como observamos anteriormente. Como resultado obtemos dois ficheiros xml com todos os veículos ordenados por quilómetros.

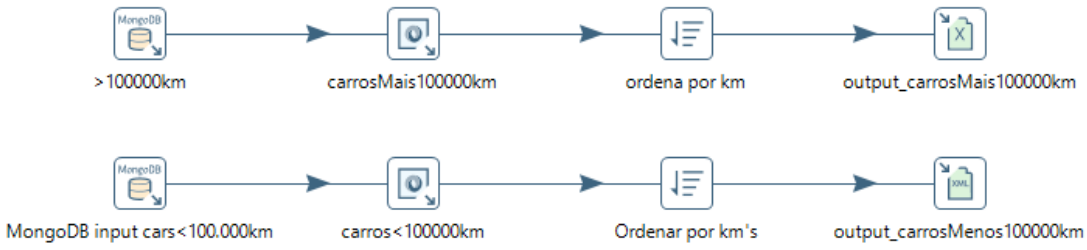

Figura 8: Entrada base dados MongoDB, saída ficheiro xml ordenado por quilómetros

Only pass unique rows? (verifies keys only) ☐

Fields :

#	Fieldname	Ascending	Case sensitive compare?	Sort based on current locale?	Collator Strength	Presorted?
1	Km	S	N	N	0	N

Figura 9: Ordenação dos veículos por respetivos quilómetros

3.3 Conversão ficheiro xml de todos os veículos com quilómetros superior a 100000km para ficheiro xls

Atendendo ao facto de um ficheiro xml não ser de leitura fácil para qualquer usuário, determinei fazer uma transformação dos dados para três ficheiros xls, onde na sua conversão os dados vão ser filtrados e separados por tipo de combustível, diesel, gasolina e elétrico, que se tornará cada um deles num ficheiro xls como mostra a figura 27.

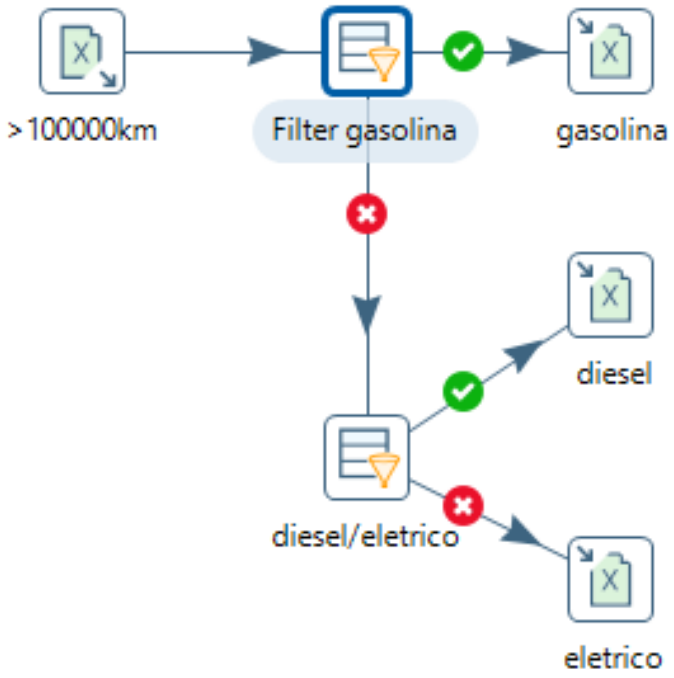

Figura 10: Entrada dados xml, filtragem dados por combustivel, resultado ficheiros xls

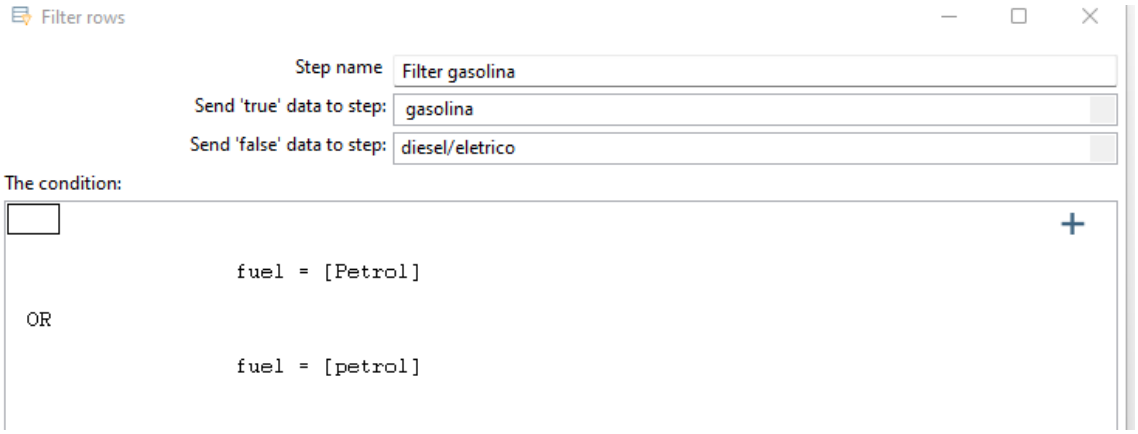

Figura 11: filtragem veículos de combustivel gasolina

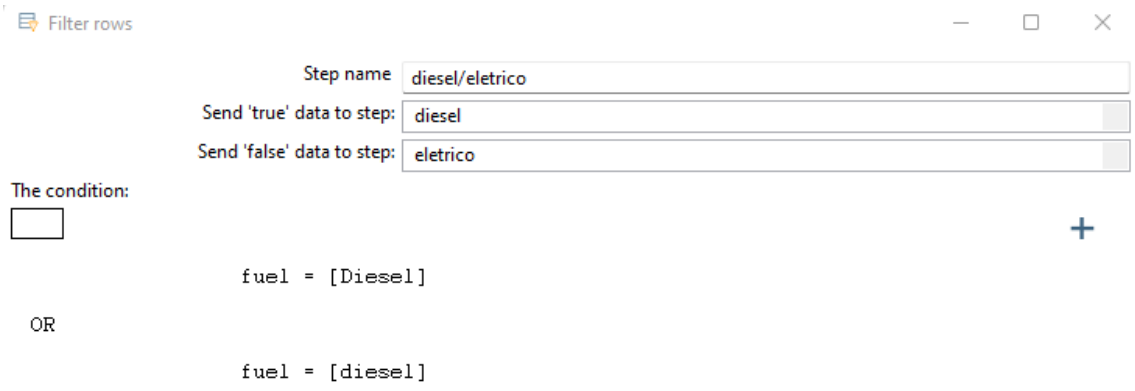

Figura 12: filtragem veículos de combustivel diesel e elétrico

3.4 Conversão ficheiro xml de todos os veículos com quilómetros inferior a 100000km para ficheiro xls

No caso anterior foi foram criados três ficheiros para armazenar os veículos de diferentes tipos de combustível, nesta abordagem que se refere aos veículos com menos de 100000km, elaboramos as mesmas filtragens, mas deste todo para "poupar" na quantidade de ficheiros de saída como resultado, vou criar apenas um ficheiro que terá na mesma todas as filtragens necessárias, e todos os veículos com menos de 100000km serão separados em cheets diferentes mas contidos no mesmo ficheiro xls.

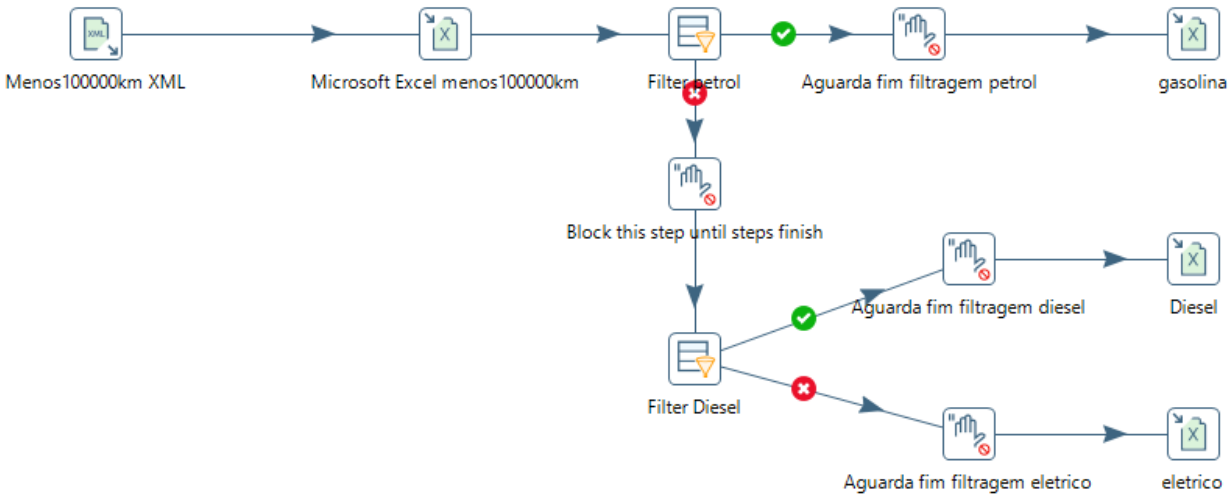

Figura 13: Entrada dados xml, filtragem dados por combustivel, resultado ficheiro xls

Para proceder á escrita dos mesmos, tive de recorrer ao Step "Block this step until steps finish", ou seja, visto que vão ser escritos no mesmo ficheiro xls a transformação deve fazer a escrita de um tipo de combustível de cada vez, esperando sempre a conclusão da escrita anterior para o ficheiro poder ser novamente acedido para modo de escrita numa nova cheet. Vejamos como exemplo a função do step da figura 27

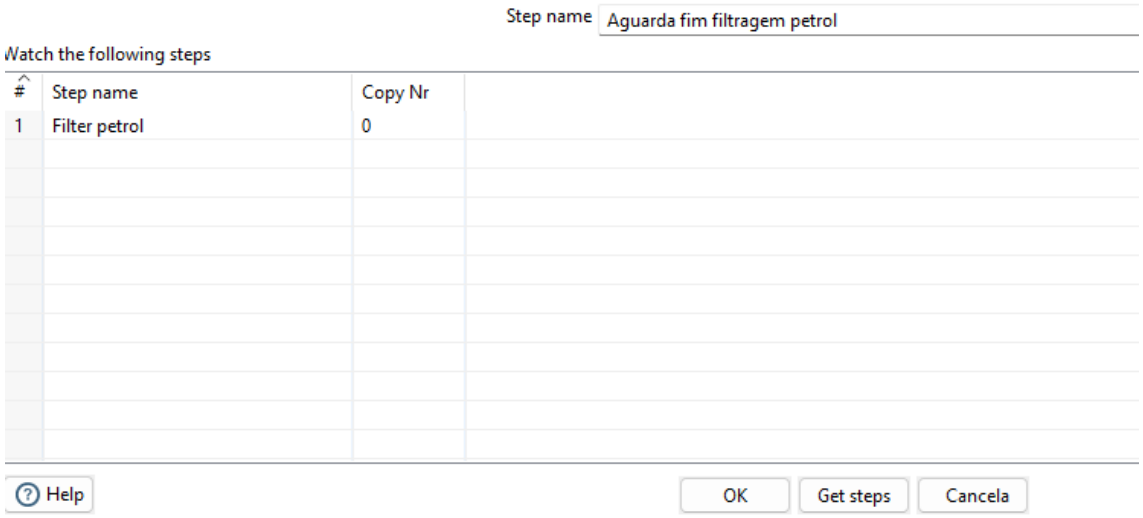

Figura 14: Aguarda o fim da filtragem de todos os veiculos a gasolina

3.5 Job

Como todas as transformações tem uma sequência e um único modo de execução visto que está sempre dependente de dados de transformações anteriores, recorri a um job, onde com apenas uma instrução fazemos sequencialmente e ordenadamente todas as transformações anteriormente faladas, obtendo eficazmente todos os resultados pretendidos com uma única instrução.

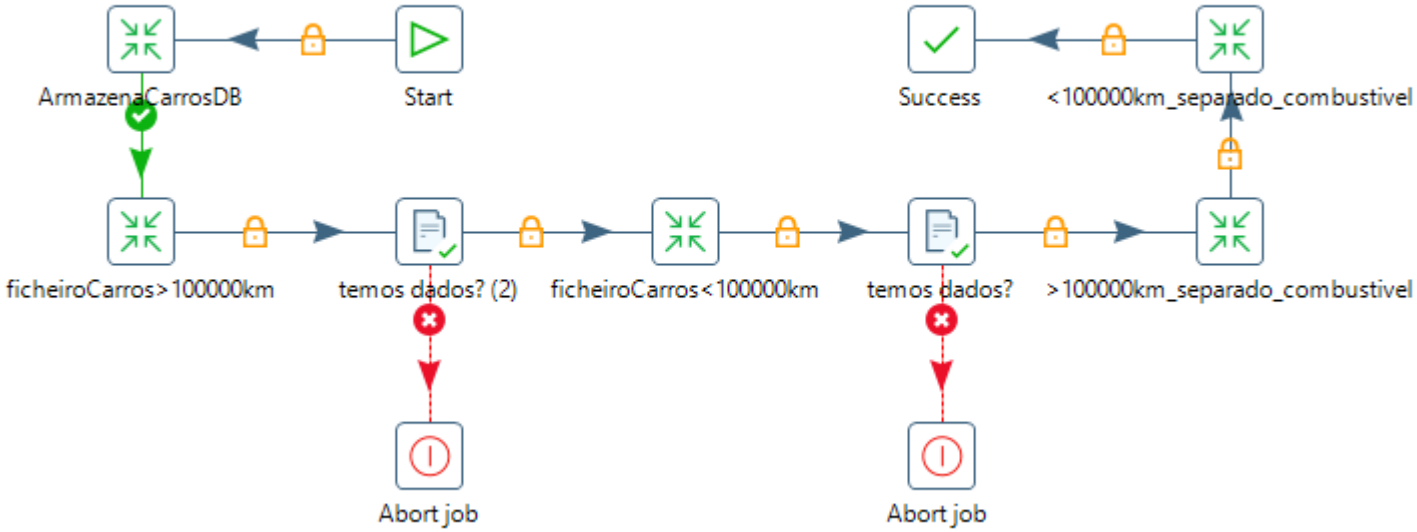

Figura 15: Job

Em cada transformação que o job executa, é feita uma verificação da existência dos dados para poder seguir a sua execução, em caso de falha de algum tipo de dados esperados, automaticamente o job deixa de fazer sentido e é então abortado com uma mensagem ao utilizador do sucedido.

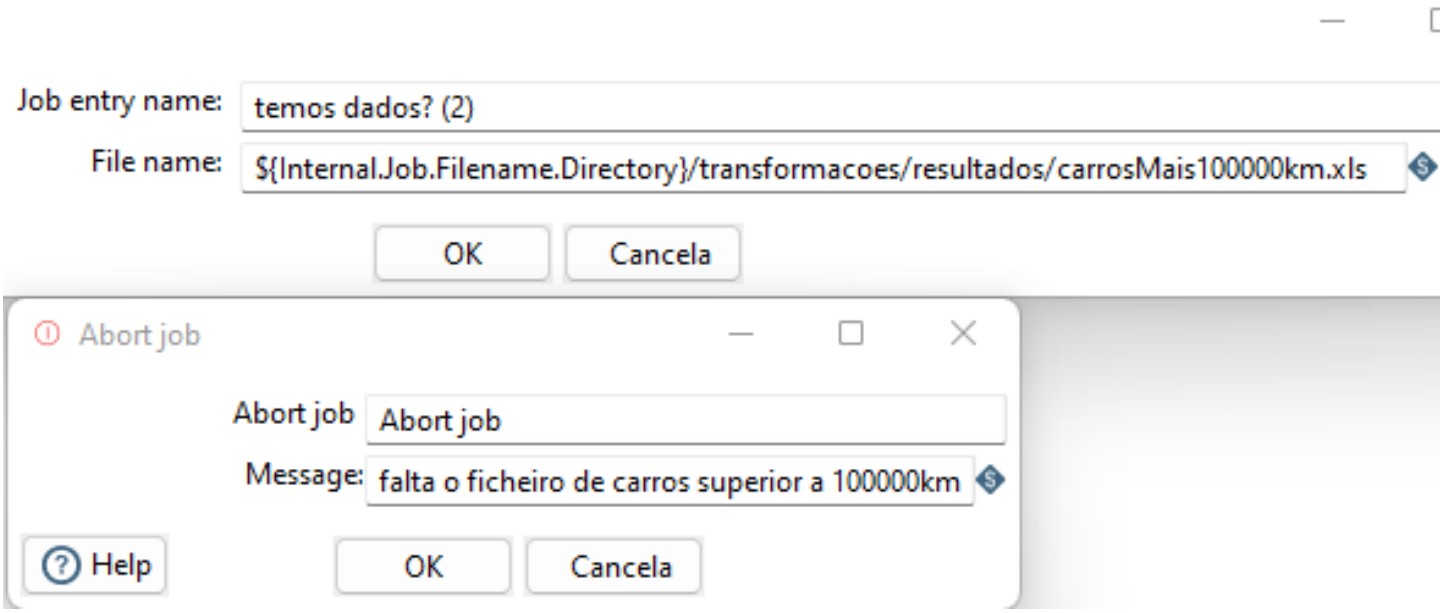

Figura 16: Confirmação da existência de dados, fim da execução no caso da condição for falsa

4 Mosquitto

4.1 Iniciação broker

O mosquito é o que chamamos de broker, ou seja, um intermediário entre máquinas e os protocolos. Ele é utilizado no protocolo MQTT para fazer com que as máquinas possam conversar entre si e agir de maneira automatizada. Neste caso usei o broker mosquitto para fazer a comunicação de dados entre o pentaho kettle e o Node-Red para que seja possível obter os dados quase em tempo real em ferramentas diferentes neste caso. Para isso usei um step do kettle chamado "MQTT Publisher" que tem como função publicar no broker localhost os dados recebidos de um json.

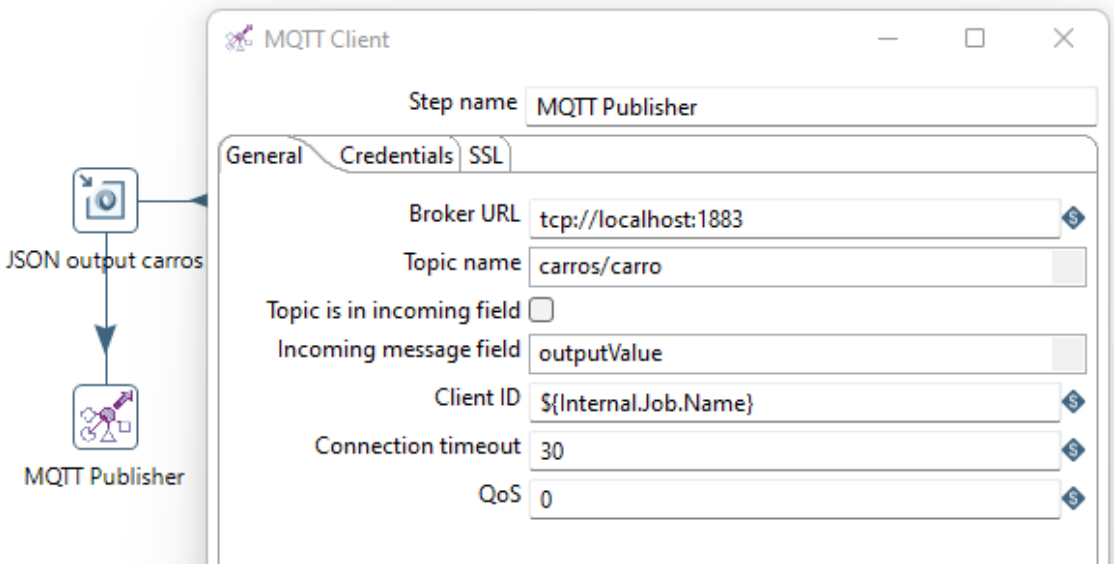

Figura 17: Step MQTT publisher kettle

Para que tal comunicação seja possivel em primeiro lugar é preciso iniciar o broker utilizando o seguinte comando:

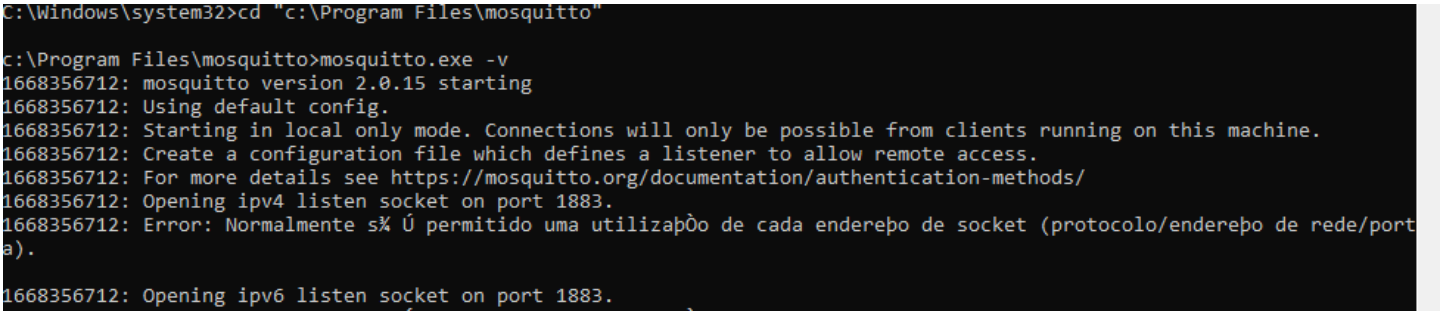

Figura 18: Iniciação do broker mosquitto

4.2 Publicação dos dados

Com o broker iniciado e o step no kettle com o topic definido é preciso um outro comando para poder então fazer a publicação dos dados no broker. o comando utilizado foi o seguinte: A

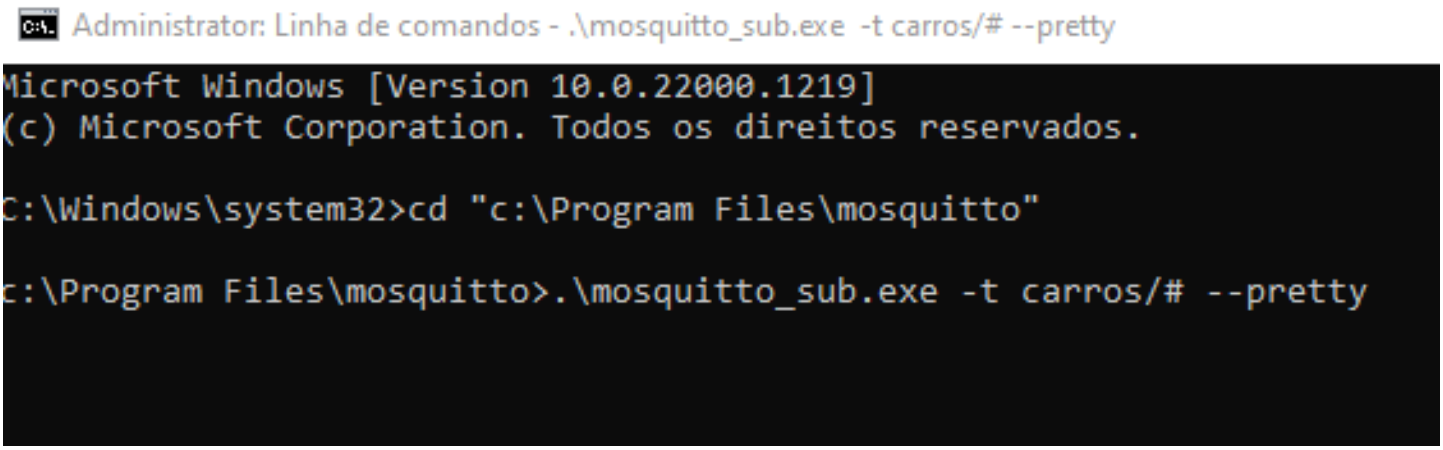

Figura 19: Iniciação do broker mosquitto

partir deste momento o broker mantém-se á escuta e quando os dados forem recebidos no json, automaticamente serão publicados no broker mosquitto e lidos no Node-Red para então poder trabalhar com os resultados obtidos do broker.

5 Node-Red

5.1 Numero total de veículos a diesel gasolina e diesel

O numero total de veículos de cada tipo de combustível, é calculado através de uma função chamada "conta numero carros Diesel, Gasolina, Elétricos", onde é incrementado um contador nos três diferentes tipos de combustível.

```
var diesel = flow.get("diesel") || 0;
var petrol = flow.get("petrol") || 0;
var eletrico = flow.get("eletrico") || 0;
if (msg.topic == "iniciar")
{
    diesel=0;
    petrol=0;
    eletrico=0;
}
else
{
    if (msg.payload.Carro[0].Fuel == "Diesel")
    {
        diesel++;
    }
    else if (msg.payload.Carro[0].Fuel == "Petrol") {
        petrol++;
    }
    else if (msg.payload.Carro[0].Fuel == "CNG") {
        eletrico++;

    }
}
flow.set("diesel", diesel);
flow.set("petrol", petrol);
flow.set("eletrico", eletrico);

msg.payload = {
    "diesel": diesel,
    "petrol": petrol,
    "eletrico":eletrico
}
return msg;
```

Figura 20: Função Javascript que conta o numero de veículos a diesel, gasolina e elétrico

Depois de receber a mensagem payload com os dados necessitamos, fazemos então a criação do gráfico de barras com a seguinte função:

```
var grafico = [{
    "data": [
        msg.payload.diesel,
        msg.payload.petrol,
        msg.payload.eletrico],

    "labels": ["Diesel", "Gasolina","Elétrico"],
    "series": [""],
}];

msg.payload = grafico;
return msg;
```

Figura 21: Função Javascript para a criação do gráfico de barras

Como resultado então obtemos:

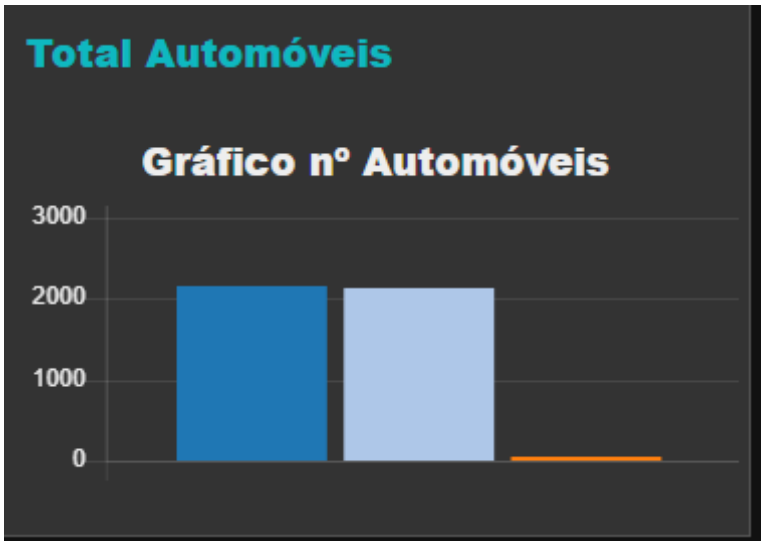

Figura 22: gráfico de barras com o numero de veículos a Diesel, Gasolina e Elétrico

5.2 Veículo com preço mais e menos elevado

Para este processo foi elaborada uma outra função chamada de ”preço mais alto/mais baixo “que é capaz de identificar o veiculo que contém o preço mais alto e o preço mais baixo de todos. Para esse efeito a função foi a seguinte:

```
var barato = flow.get("barato") || msg.payload.Carro[0].Preco;
var caro = flow.get("caro") || msg.payload.Carro[0].Preco;
if(msg.topic == "iniciar")
{
    barato=0;
    caro=0;
}else
{
    if (msg.payload.Carro[0].Preco >= caro) {
        caro = msg.payload.Carro[0].Preco
    }
    else if (msg.payload.Carro[0].Preco <= barato) {
        barato = msg.payload.Carro[0].Preco
    }
    flow.set("caro", caro)
    flow.set("barato", barato)



}
msg.payload = {
    "caro": caro,
    "barato": barato
}

return msg;
```

Figura 23: função que determina o veiculo com preço mais e menos elevado

Como resultado obtemos:

Figura 24: função que determina o veiculo com preço mais e menos elevado

5.3 Nome do veículo com preço mais e menos elevado

Também achei importante ter ao alcance o nome do veículo com o preço mais e menos elevado, para poder servir de informação a qualquer tipo de cliente. Para tal usei a seguinte função:

```
var barato = flow.get("barato") || msg.payload.Carro[0].Preco;
var caro = flow.get("caro") || msg.payload.Carro[0].Preco;
if (msg.topic == "iniciar") {
    nomeBarato = "não encontrado"
    nomeCaro = "não encontrado"
}else{
    if (msg.payload.Carro[0].Preco >= caro) {
        caro = msg.payload.Carro[0].Preco
    }
    else if (msg.payload.Carro[0].Preco <= barato) {
        barato = msg.payload.Carro[0].Preco
    }

    flow.set("caro", caro)
    flow.set("barato", barato)
    if (msg.payload.Carro[0].Preco >= caro) {
        caro = msg.payload.Carro[0].Preco
    }
    else if (msg.payload.Carro[0].Preco <= barato) {
        barato = msg.payload.Carro[0].Preco
    }

    flow.set("caro", caro)
    flow.set("barato", barato)


    if (msg.payload.Carro[0].Preco == precobarato) {
        nomeBarato = msg.payload.Carro[0].Nome;

    }
    else if (msg.payload.Carro[0].Preco == precocaro) {
        nomeCaro = msg.payload.Carro[0].Nome;

    }

}




flow.set("nome1", nomeBarato);
flow.set("nome2", nomeCaro);

msg.payload = {
    "nome1": nomeBarato,
    "nome2": nomeCaro,
}

return msg;
```

Figura 25: função que determina o nome do veículo com preço mais e menos elevado

Como resultado final obtemos então:

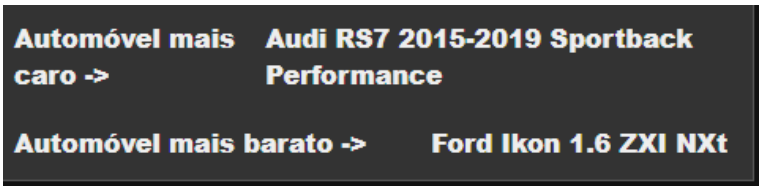

Figura 26: Nome dos veículos com preço mais e menos elevado

5.4 Flow

Para dar resposta a todos estes resultados o meu flow teve o seguinte resultado seguinte:

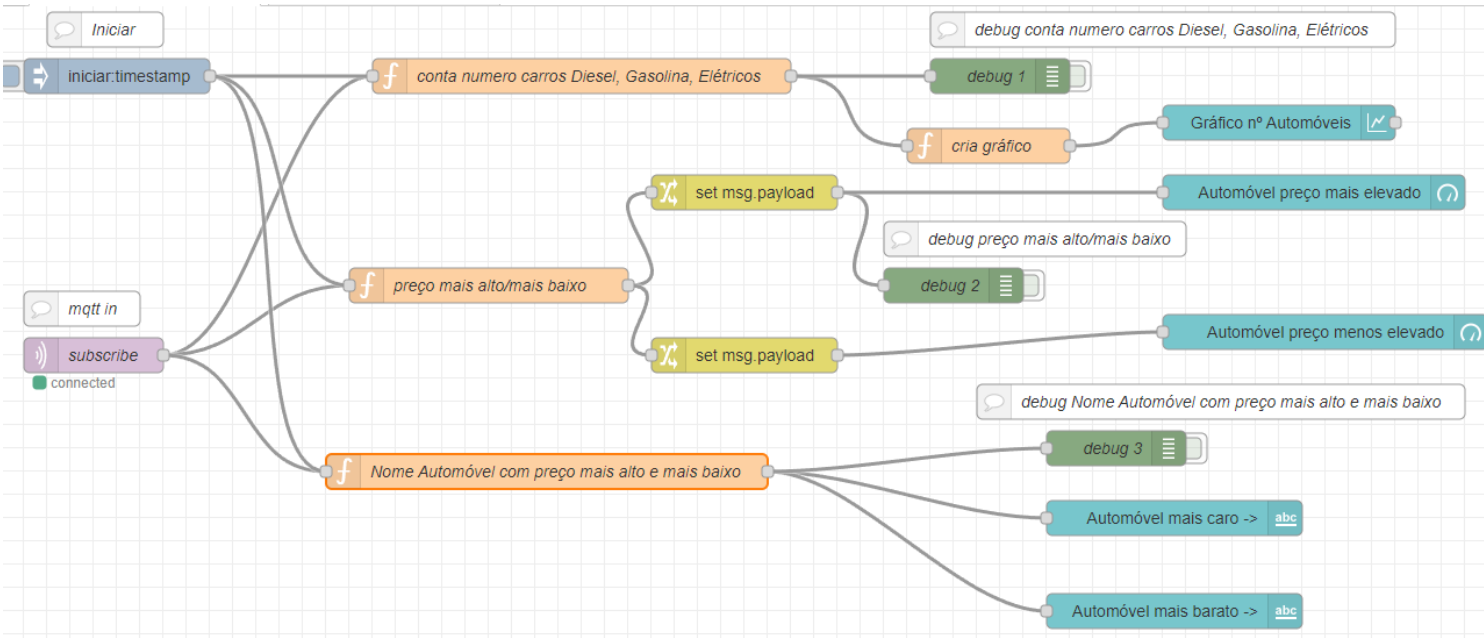

Figura 27: Flow

6 Conclusão

De uma forma geral, fiquei satisfeito com o trabalho desenvolvido.

Senti algumas dificuldades na escolha do meu tema, mas foi o mesmo que me ajudou a interpretar estas poderosas ferramentas que foram abordadas.

Durante o meu trabalho, senti algumas dificuldades na interpretação da ferramenta Node Red, mas felizmente consegui desenvolver o que pretendia para este projeto.

Tenho consciência que podia ter implementado melhor o enunciado pretendido, mas infelizmente não tive tempo para mais.

Contudo, concluo que a estrutura do trabalho implementada foi útil para a compreensão e interiorização dos conteúdos lecionados durante a unidade curricular de Integração de sistemas de Informação.

7 Referências

- http://holowczak.com/building-etl-transformations-in-pentaho-data-integration-kettle/
- https://docs.genesys.com/Documentation/IWD/latest/Ref/IWDUsingKettle
- http://www.steves-internet-guide.com/mosquitto-broker/
- https://ncd.io/connecting-the-mqtt-gateway-to-mosquitto/
- https://nodered.org
- https://flows.nodered.org/node/node-red-dashboard
- https://www.digikey.com/en/maker/blogs/2022/create-user-dashboards-for-iot-projects-in-node-red