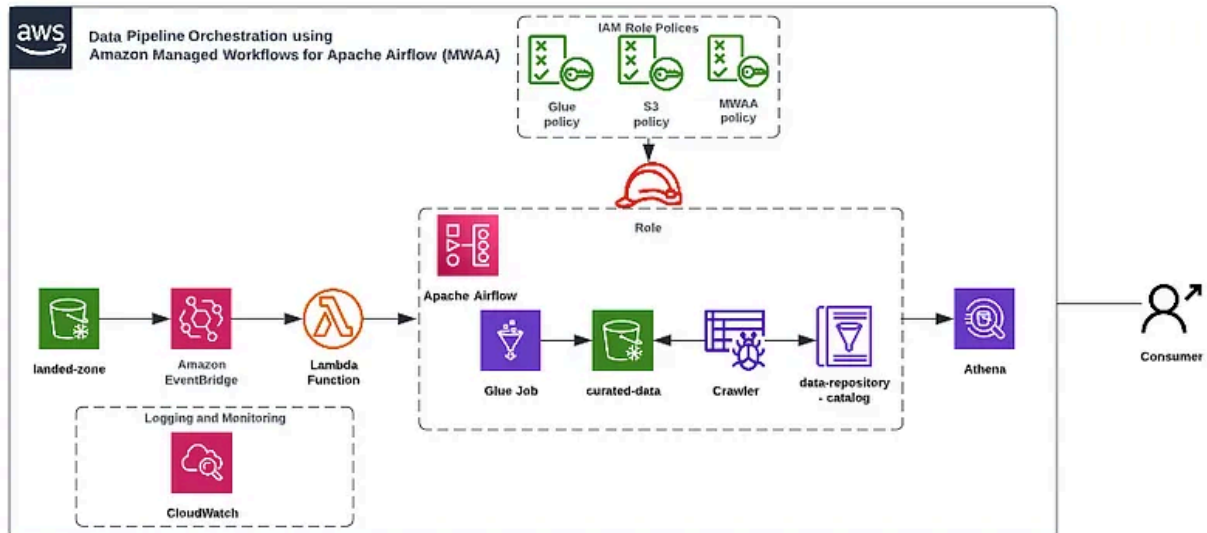


Fuente:

<https://medium.com/contino-engineering/data-pipeline-orchestration-using-amazon-managed-workflows-for-apache-airflow-mwaa-60e5b213a0a7>

## Architecture

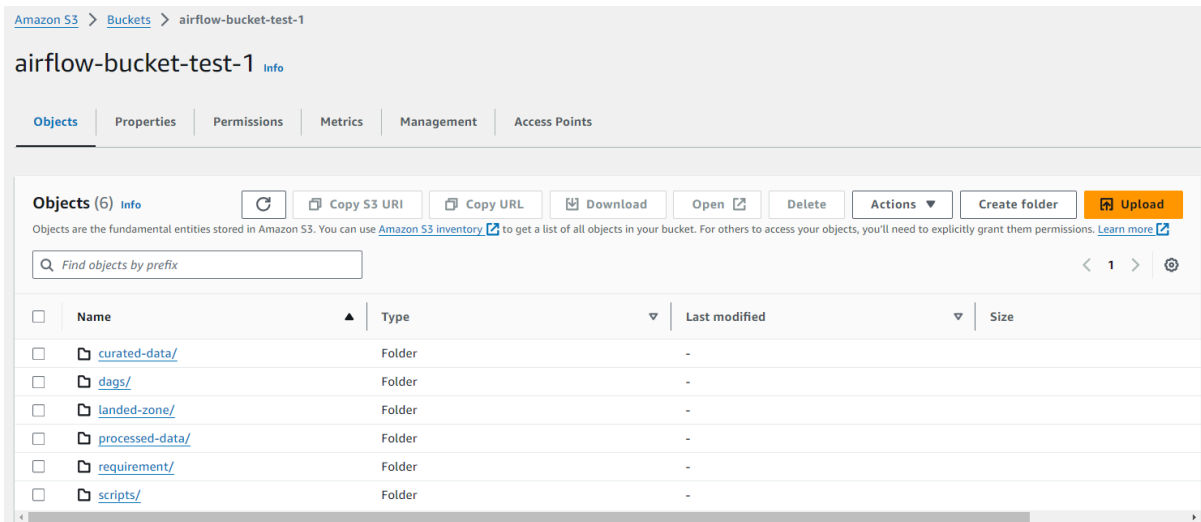


## Process Overview

1. Create an S3 bucket using GUI.
2. Configure Amazon Managed Workflow for Apache Airflow (MWAA).
  - IAM Role: Create IAM roles and policies to use with MWAA.
  - S3 Bucket: S3 bucket to store your DAGs, plugins, and other Airflow files.
  - VPC and Subnet: To deploy an MWAA environment in a VPC and Subnet for secure and private network communication.
  - Security Groups: Create security groups to control MWAA environment access.
  - Monitoring and logging: Configure Amazon CloudWatch to monitor and log your MWAA environment.
3. Create the following AWS artifacts using the CloudFormation template:
  - IAM Role: Grant permission to AWS Glue and S3 services and attach this role with the AWS Glue job.
  - Glue Job: Converting CSV file to Parquet format and saving the curated file(s) into S3.
  - Crawler: Crawl and Catalog curated data using AWS Glue Crawler.
  - Catalog: Catalog the metadata of the process file.
4. Lambda function to trigger MWAA pipeline.
5. Upload the CSV data files to S3 (Landing Zone) bucket.
6. Query transformed data using AWS Athena.

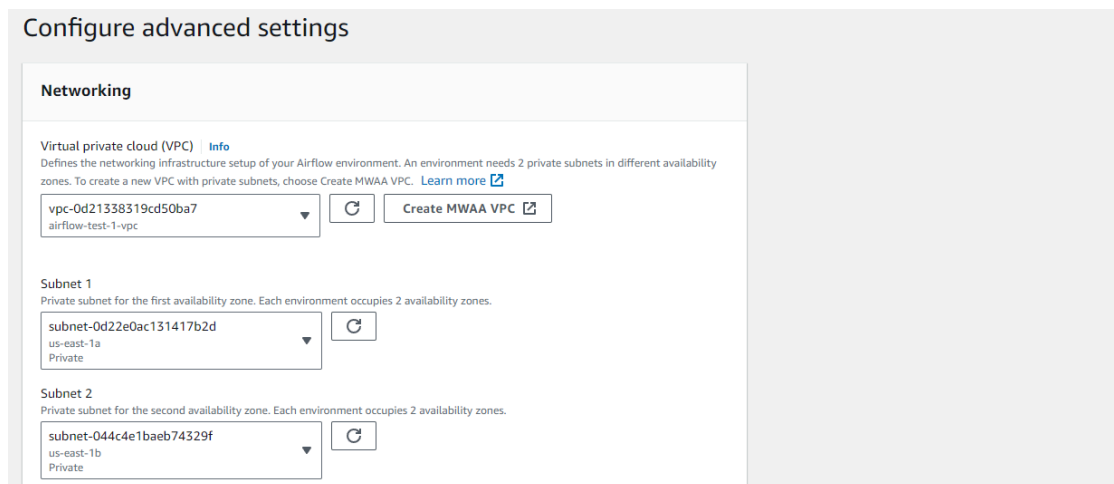
Creamos un bucket en S3 con la configuración básica. Luego agregamos los siguientes folders:

- curated-data: Store curated data in this folder.
- dags: Store DAGs code.
- landed-zone: Contains the raw data file to process.
- scripts: To save AWS Glue job script(s).
- processed-data: To move the processed raw data files.
- requirement: Contains DAG “requirement.txt” file.



Vamos al servicio MWAA y creamos un nuevo environment con la siguiente configuración:

- Asignamos un nombre y versión (última).
- Seleccionamos el bucket de s3 creado anteriormente y apuntamos a la carpeta dags.
- Creamos una VPC o elegimos una ya creada. Debe haber al menos 2 subnets privadas para seleccionarlal en la configuración.



- Elegimos un acceso público a través de internet.

Web server access [Info](#)

☐ Private network (No internet access)

Additional setup required. Your Airflow UI can only be accessed by secure login behind your VPC. Choose this option if your Airflow UI is only accessed within a corporate network and you do not require a public repository for webserver requirements installation. IAM must be used to handle user authentication.

☒ Public network (Internet accessible)

Your Airflow UI can be accessed by secure login over the Internet. Choose this option if your Airflow UI is accessed outside of a corporate network. IAM must be used to handle user authentication.

- Elegimos un security group o creamos uno desde cero. En este caso ya tenía uno creado.
- Elegimos un IAM role (para MAAA) o creamos uno desde cero. En este caso ya tenía uno creado.
- La configuración restante se deja por defecto.

*Esperamos de 20 a 30 minutos a que se cree el environment.*

Creamos los demás servicios con apoyo del servicio CloudFormation, utilizando el template del tutorial.

- IAM Glue role
- Glue crawler
- Glue data catalog
- Glue job

***Dar el acceso completo a S3 (AmazonS3FullAccess) no es recomendable. Se deja así solo para el tutorial.***

Elegimos el template que ya está en los archivos locales.

Create stack

Prerequisite - Prepare template

You can also create a template by scanning your existing resources in the [laC generator](#)

Prepare template

Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

☒ Choose an existing template
Upload or choose an existing template.

☐ Use a sample template
Choose from our sample template library.

☐ Build from Application Composer
Create a template using a visual builder.

Specify template [Info](#)

A template is a JSON or YAML file that describes your stack's resources and properties.

Template source

Selecting a template generates an Amazon S3 URL where it will be stored.

☐ Amazon S3 URL
Provide an Amazon S3 URL to your template.

☒ Upload a template file
Upload your template directly to the console.

☐ Sync from Git - new
Sync a template from your Git repository.

Upload a template file

Choose file

stack.yml

JSON or YAML formatted file

S3 URL: <https://s3.us-east-1.amazonaws.com/cf-templates-fkrreh3lx4ta-us-east-1/2024-08-28T161055.5302pny-stack.yml>

View in Application Composer

Al dar click en next nos aparecen los parámetros definidos en el template.

Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

CrawlerS3Path

Target file name for AWS Crawler

s3://airflow-bucket-test-1/curated-data/

GlueCatalogName

Name of the data Catalog database

curated-data

GlueCrawlerName

Glue Crawler name to crawl the raw data

catalog-cars-data

GlueJobName

Glue Job Name to process the raw data

cars-data-transformation

GlueJobScriptLocation

Glue job script location

s3://airflow-bucket-test-1/scripts/etlscript.py

IAMRoleName

Name of the existing role

demo-mwaa-glue

El stack de CloudFormation se ejecuta satisfactoriamente.

CloudFormation > Stacks > stack-airflow-1

Stacks (1)

Filter by stack name

Filter status

Active

View nested

Stacks

stack-airflow-1

2024-08-28 11:21:59 UTC-0500

CREATE\_COMPLETE

stack-airflow-1

Delete

Update

Stack actions

Create stack

Stack info

Events

Resources

Outputs

Parameters

Template

Change sets

Git sync - new

Events (14)

Detect root cause

Timestamp	Logical ID	Status	Detailed status	Status reason
2024-08-28 11:22:23 UTC-0500	stack-airflow-1	CREATE_COMPLETE	-	-
2024-08-28 11:22:22 UTC-0500	GlueCrawler	CREATE_COMPLETE	-	-
2024-08-28 11:22:22 UTC-0500	GlueCrawler	CREATE_IN_PROGRESS	-	Resource creation Initiated
2024-08-28 11:22:21 UTC-0500	GlueCrawler	CREATE_IN_PROGRESS	-	-
2024-08-28 11:22:20 UTC-0500	IAMRole	CREATE_COMPLETE	-	-
2024-08-28 11:22:04 UTC-0500	GlueCatalog	CREATE_COMPLETE	-	-
2024-08-28 11:22:03 UTC-0500	GlueJob	CREATE_COMPLETE	-	-
2024-08-28 11:22:03 UTC-0500	GlueJob	CREATE_IN_PROGRESS	-	Resource creation Initiated
2024-08-28 11:22:03 UTC-0500	GlueCatalog	CREATE_IN_PROGRESS	-	Resource creation Initiated
2024-08-28 11:22:03 UTC-0500	IAMRole	CREATE_IN_PROGRESS	-	Resource creation Initiated
2024-08-28 11:22:02 UTC-0500	IAMRole	CREATE_IN_PROGRESS	-	-
2024-08-28 11:22:02 UTC-0500	GlueCatalog	CREATE_IN_PROGRESS	-	-

Verificamos el environment de Airflow al que ya se puede acceder.

Airflow

DAGs

Cluster Activity

Datasets

Security

Browse

Admin

Docs

16:28 UTC

DAGs

Active

Paused

Running

Failed

Filter DAGs by tag

Search DAGs

DAG

Owner

Runs

Schedule

Last Run

Next Run

Recent Tasks

Actions

Links

No results

Showing 0-0 of 0 DAGs

Ahora creamos la función Lambda.

Basic information

Function name

Enter a name that describes the purpose of your function.

function-mwaa-pipeline

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime

Info

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Python 3.12

Architecture

Info

Choose the instruction set architecture you want for your function code.

x86\_64

arm64

Permissions

Info

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console

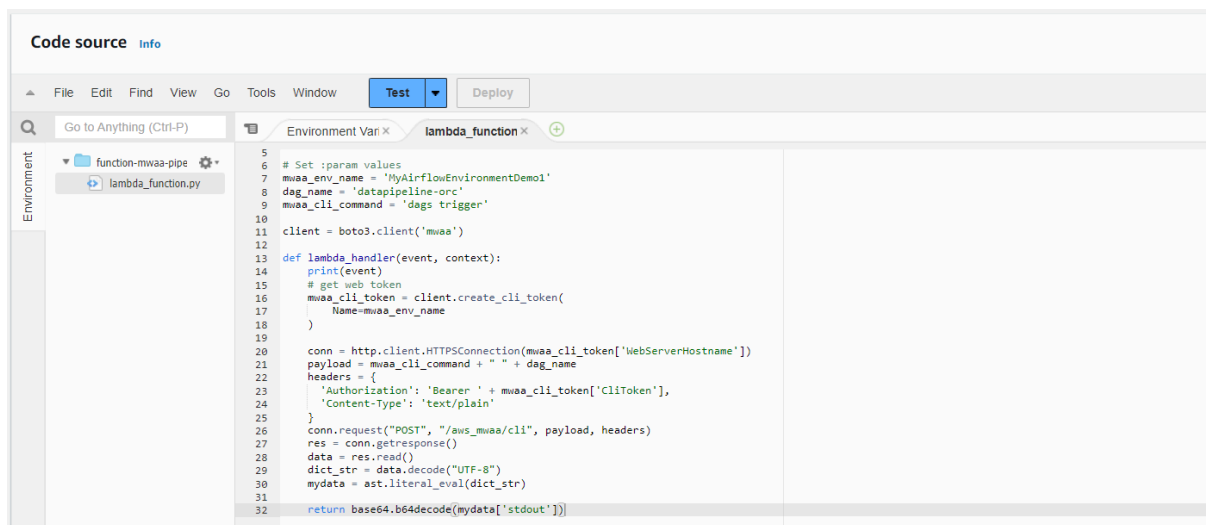
Create a new role with basic Lambda permissions

Use an existing role

Create a new role from AWS policy templates

No apareció el rol que se creó para el environment de MWAA, por lo que creé uno nuevo con los permisos básicos de Lambda.

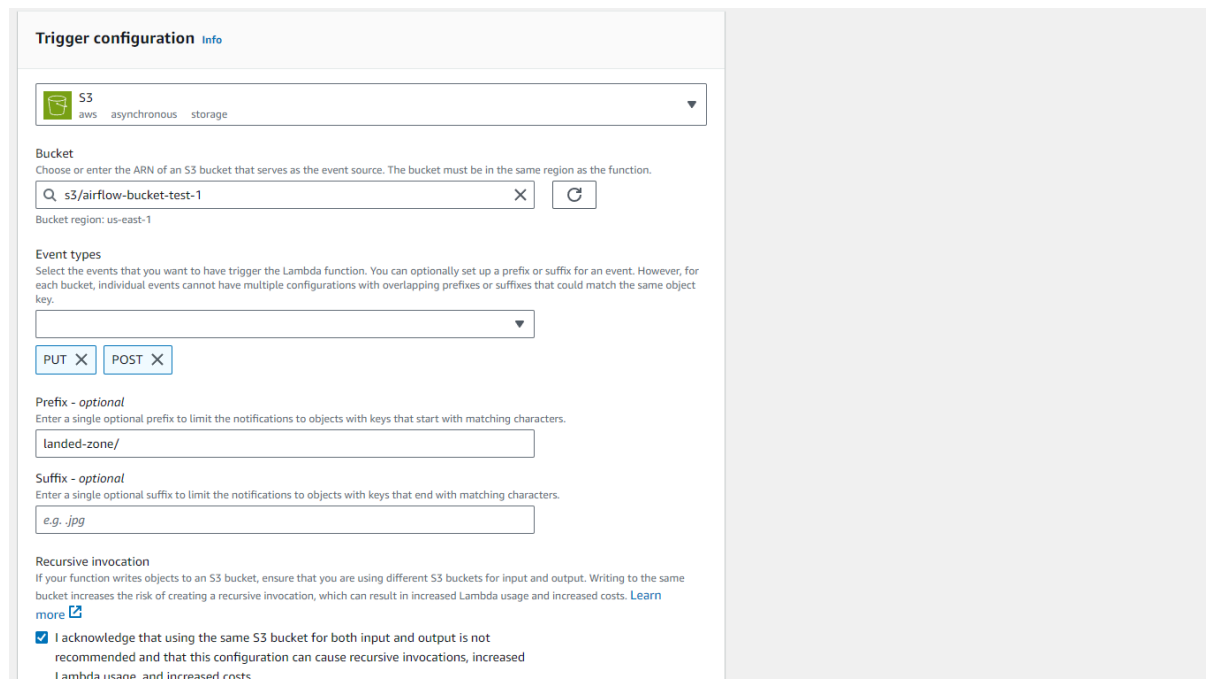
Agregamos el código proporcionado en el tutorial.



The screenshot shows the AWS Lambda console's 'Code source' tab for a function named 'lambda\_function'. The code is a Python script that interacts with AWS services. It sets environment variables for 'mwaa\_env\_name', 'dag\_name', and 'mwaa\_cli\_command'. It then uses the boto3 library to create a client for the 'mwaa' service. The 'lambda\_handler' function is defined, which prints the event, gets a web token, creates a CLI token, and makes a POST request to the 'aws\_mwaa/cli' endpoint. The response is decoded and returned as a base64-encoded string.

```
5 # Set :param values
6 mwaa_env_name = 'MyAirflowEnvironmentDemo1'
7 dag_name = 'datapipeline-orc'
8 mwaa_cli_command = 'dags trigger'
9
10
11 client = boto3.client('mwaa')
12
13 def lambda_handler(event, context):
14     print(event)
15     # get web token
16     mwaa_cli_token = client.create_cli_token(
17         Name=mwaa_env_name
18     )
19
20     conn = http.client.HTTPSConnection(mwaa_cli_token['WebServerHostname'])
21     payload = mwaa_cli_command + " " + dag_name
22     headers = {
23         'Authorization': 'Bearer ' + mwaa_cli_token['CliToken'],
24         'Content-Type': 'text/plain'
25     }
26     conn.request("POST", "/aws_mwaa/cli", payload, headers)
27     res = conn.getresponse()
28     data = res.read()
29     dict_str = data.decode("UTF-8")
30     mydata = ast.literal_eval(dict_str)
31
32     return base64.b64decode(mydata['stdout'])
```

Agregamos un trigger a la función, apuntando al bucket de S3 al presentarse un evento post o put.



The screenshot shows the 'Trigger configuration' tab in the AWS Lambda console. It is configured to be triggered by an S3 bucket named 's3/airflow-bucket-test-1' in the 'us-east-1' region. The event types are set to 'PUT' and 'POST'. The prefix is 'landed-zone/' and the suffix is '.jpg'. The 'Recursive invocation' checkbox is checked, indicating that the function can be triggered recursively by writing to the same bucket.

**Trigger configuration** info

**S3**  
aws asynchronous storage

**Bucket**  
Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.  
s3/airflow-bucket-test-1  
Bucket region: us-east-1

**Event types**  
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.  
PUT POST

**Prefix - optional**  
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.  
landed-zone/

**Suffix - optional**  
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.  
e.g. .jpg

**Recursive invocation**  
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. [Learn more](#)

☒ I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

En este punto debemos subir el script del DAG al folder que tenemos en el bucket de S3, el cual contiene la siguientes tareas (tasks):

- begin: Starting the DAG point
- purge\_data\_catalog: Delete the data catalog
- purge\_processed\_data\_s3\_objects: Clean up the processed data folder in the S3 bucket
- run\_glue\_job: Trigger glue job to convert raw data file format from CSV to PARQUET
- run\_glue\_crawler: Crawler the curated-data folder and catalog the metadata of the processed file(s)
- sync\_buckets: Copy landed-zone bucket objects to the processed-data folder
- purge\_raw\_data\_file: Clean up the landed-zone folder
- end: Show Data pipeline orchestration completion



Cargamos de nuevo la interface y nos aparece el DAG.

Damos click en el interruptor que se encuentra al lado del nombre de nuestro DAG, para dejarlo como *activo*.

## DAGs

Algo que no se mencionó en el tutorial fue cargar el script del ETL de Glue en la carpeta de scripts del bucket de S3.

Cargamos la data (carsdata.csv) al folder landed-zone para activar la función Lambda y que esta a su vez inicie la ejecución del DAG en Airflow.

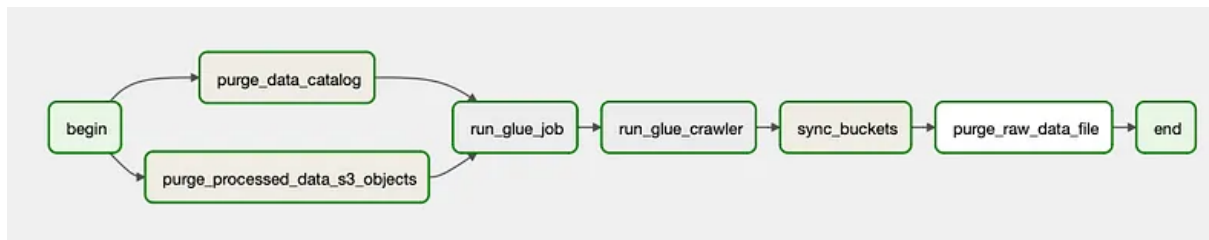
Se tuvo que añadir la action `CreateCliToken` al rol de `MWAA` y atar esta policy al rol de `Lambda`.

```

{
  "Effect": "Allow",
  "Action": [
    "airflow:PublishMetrics",
    "airflow:CreateCliToken"
  ],
  "Resource": "arn:aws:airflow:us-east-1:183036207239:environment/MyAirflowEnvironment1"
},

```

Luego termina de correr el DAG.



En este punto ya es posible ir hasta Athena y ver qué hay en la tabla curated\_data.

1 `SELECT * FROM "curated-data"."curated_data" limit 10;`

SQL Ln 1, Col 1

[Run again](#) [Explain](#) [Cancel](#) [Clear](#) [Create](#)

☐ Reuse query results  
\*Athena engine version 3 only

[Query results](#) | [Query stats](#)

Completed Time in queue: 217 ms Run time: 434 ms Data scanned: 0.47 KB

Results (10) [Copy](#) [Download results](#)

Search rows

#	make	model	year	price
1	Honda	CVR	2012	8000
2	Toyota	Corolla	2012	7000
3	BMW	X5	2023	8900
4	Acura	mDX	2022	6788
5	Honda	CVR	2012	8000