Deloitte Take Home Challenge

Senthil Esakkiappan
08/18/2023

Tools

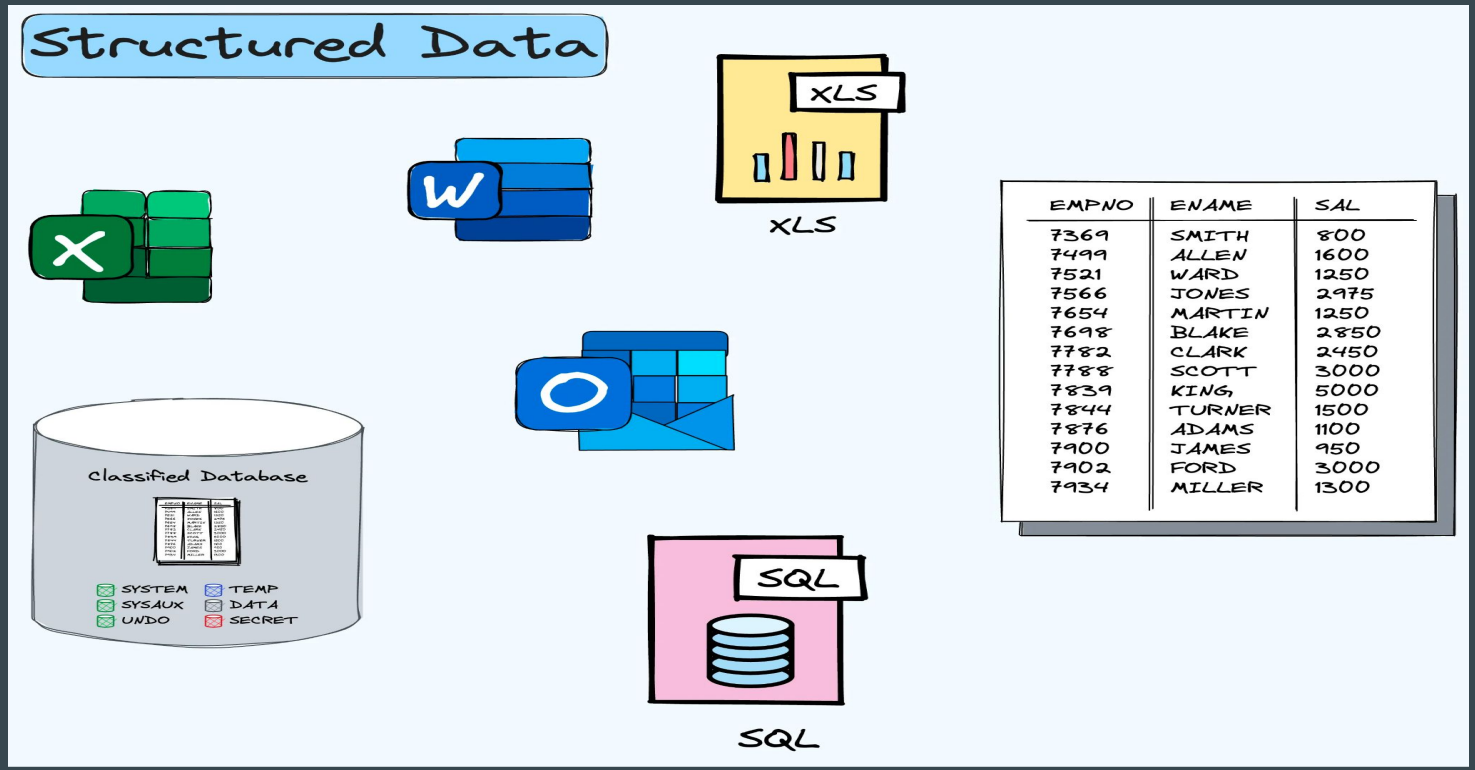
- All the visualizations are created using Excalidraw
- All the mindmap diagrams are created using Gitmind
- For IDE - Pycharm and Google Colab

What is a Data Lake?

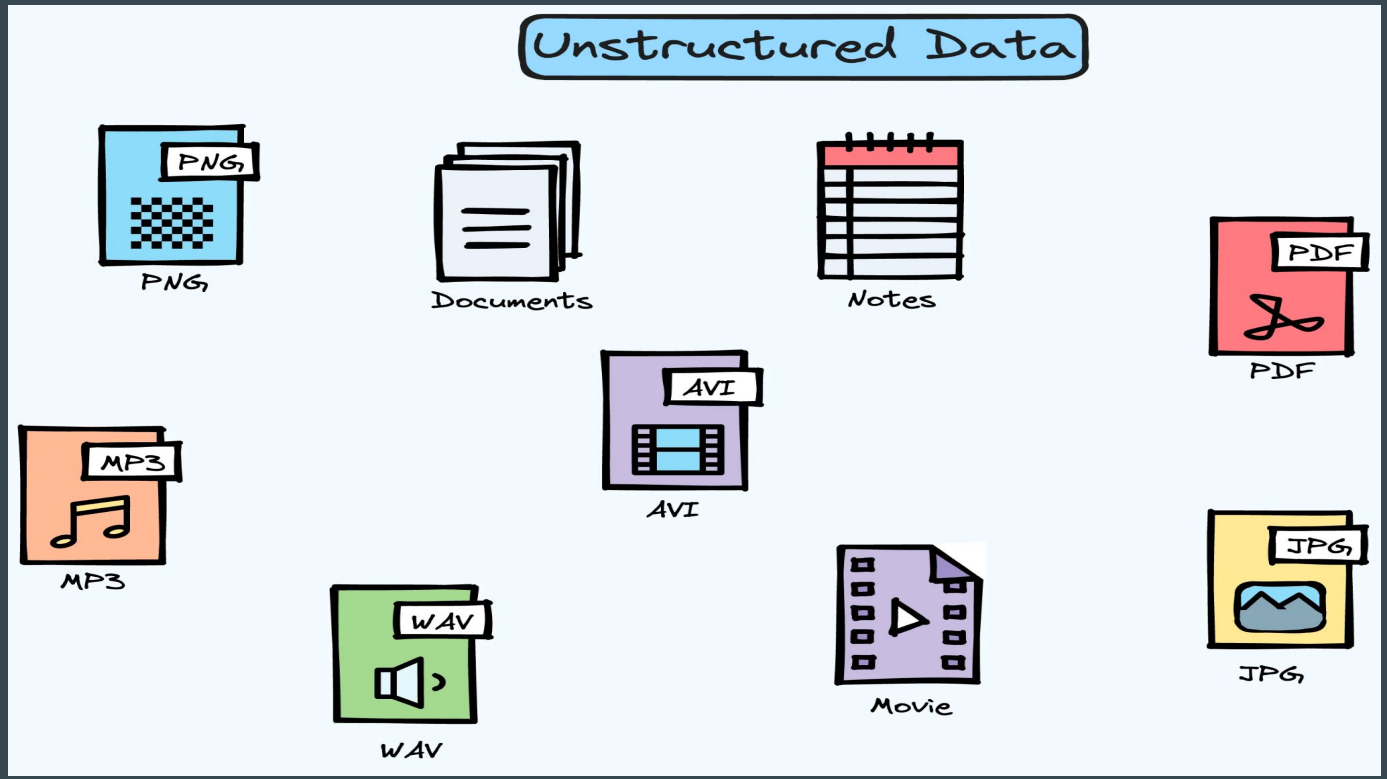
- A Data Lake is like a big storage pool where businesses can pour in all their raw data. This data can be in any format, like documents, videos, databases, or spreadsheets.

- A data lake is a centralized repository that allows you to store all your structured and unstructured data at any scale.

Structured Data



Data Lake


Unstructured Data
PNG
PNG
Documents
Notes
PDF
PDF
AVI
AVI
MP3
MP3
WAV
WAV
Movie
JPG
JPG

Unstructured Data

It does not have a pre-defined format or pre-defined schema.

It's called "unstructured" because it doesn't fit into tables or rows like structured data.

Some examples are text documents, emails, videos, audio files, social media posts, satellite images, etc.

Unstructured data accounts for more than 80% of all data and is part of the big data. Source: IDC estimates & Gartner.

It is stored in data lakes, object storage, or specialized databases like NoSQL, time-series, or vector databases

Benefits of a Data Lake:

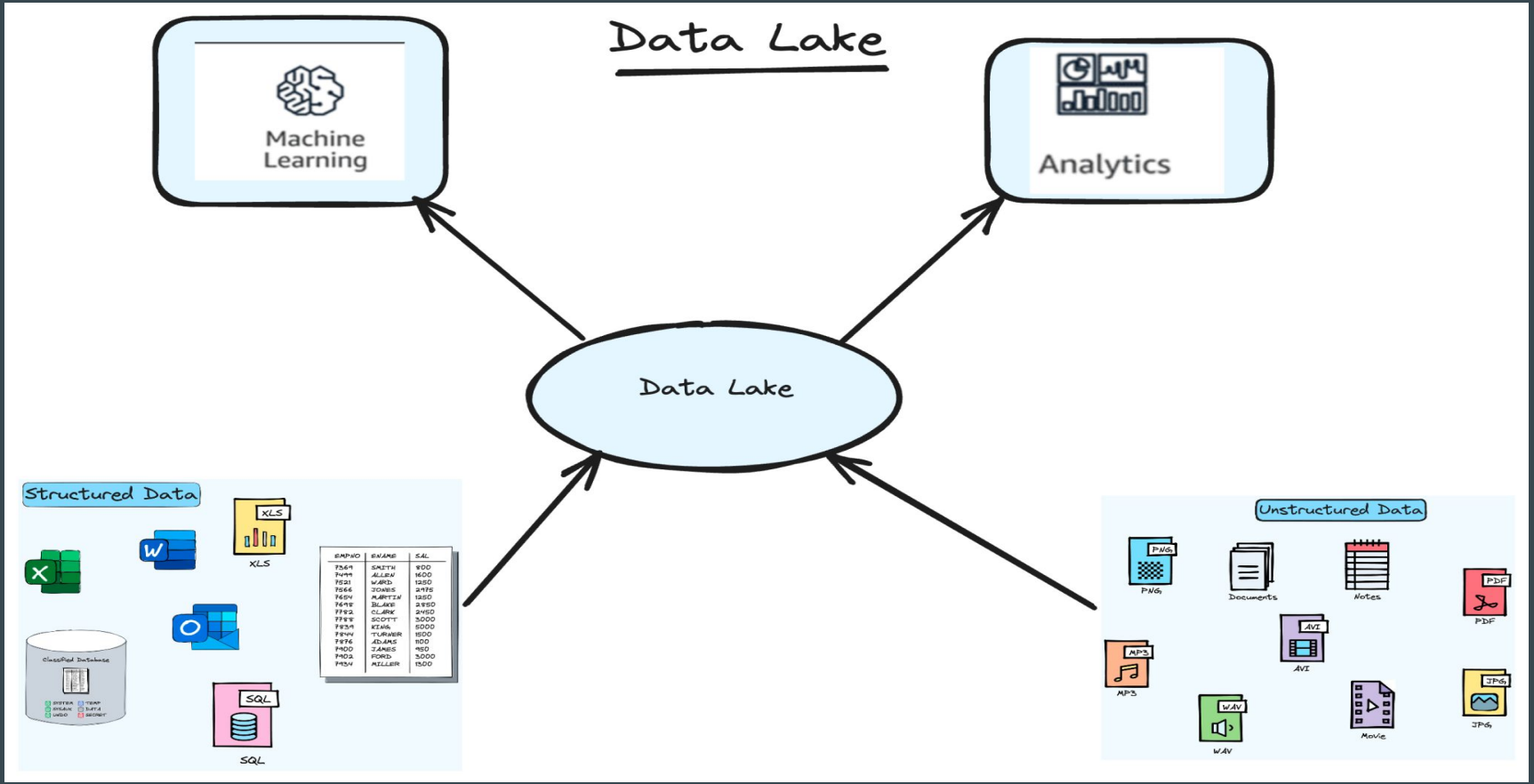
Variety of Data: You can store any type of data, whether it's structured (like databases) or unstructured (like videos).

Scale: It's built to store a lot of data, and can expand easily.

Flexibility: You don't need to know exactly how you'll use the data when you store it. You can decide later and then process and analyze it in different ways.

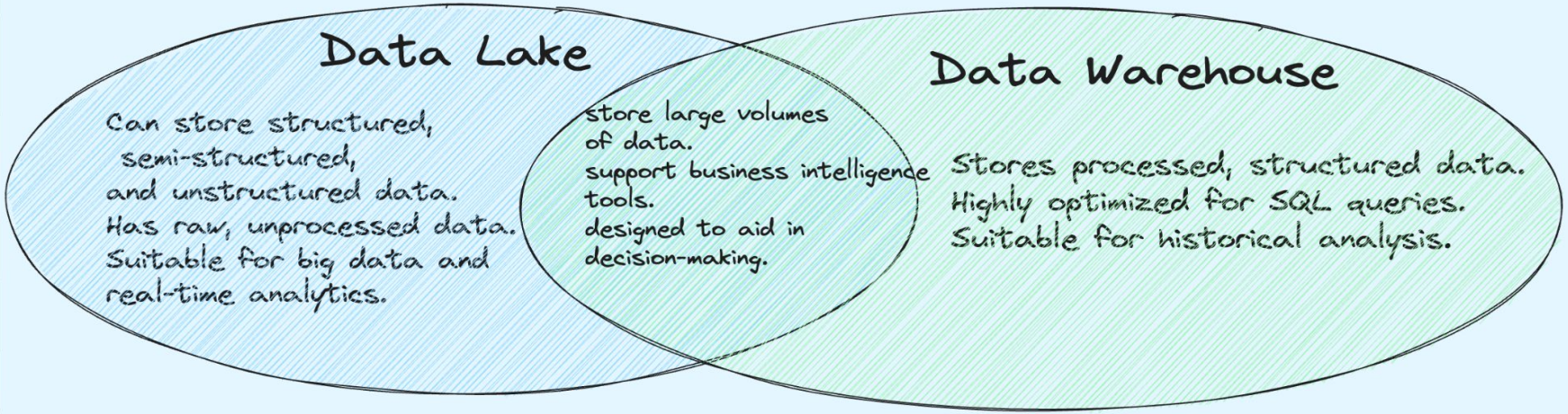
Cost-effective: It's usually cheaper to store large amounts of raw data in a data lake compared to other storage solutions.

Data Lake


Data Lake
Machine Learning
Analytics
Data Lake
Structured Data
XLS
SQL
Unstructured Data
PNG
Documents
Notes
PDF
AVI
MP3
WAV
Movie
JPG

How is a Data Lake different from a Data Warehouse?

A	B	C
Feature	**Data Lake**	**Data Warehouse**
Type of Data	Raw and unstructured (e.g., videos, logs)	Processed and structured (e.g., tables)
Purpose	Storage of vast amounts of raw data	Organized storage for analysis
Cost	Typically cheaper for storing raw data	Can be costlier due to processing
Scalability	Highly scalable	Scalable, but may require more resources
Processing Time	Data is processed when read	Data is processed before storage
Users	Data scientists, analysts	Business analysts, end users
Query Performance	Can be slower (as data is raw)	Faster (as data is organized)
Schema	Schema-on-read (define structure when read)	Schema-on-write (structure when stored)
Integration	Can integrate various types and sources	Primarily integrates structured sources



Data Lake Vs Data Warehouse

Benefits to the Client:

Future Proof: Since clients can store all kinds of data without worrying about structure, they're prepared for future needs. They won't regret not saving something important.

Insights and Discovery: With all their data in one place, they can use tools to discover new patterns and insights.

Flexibility: If clients want to test out new ways of analyzing data, they have all their raw data ready to play with in the Data Lake.

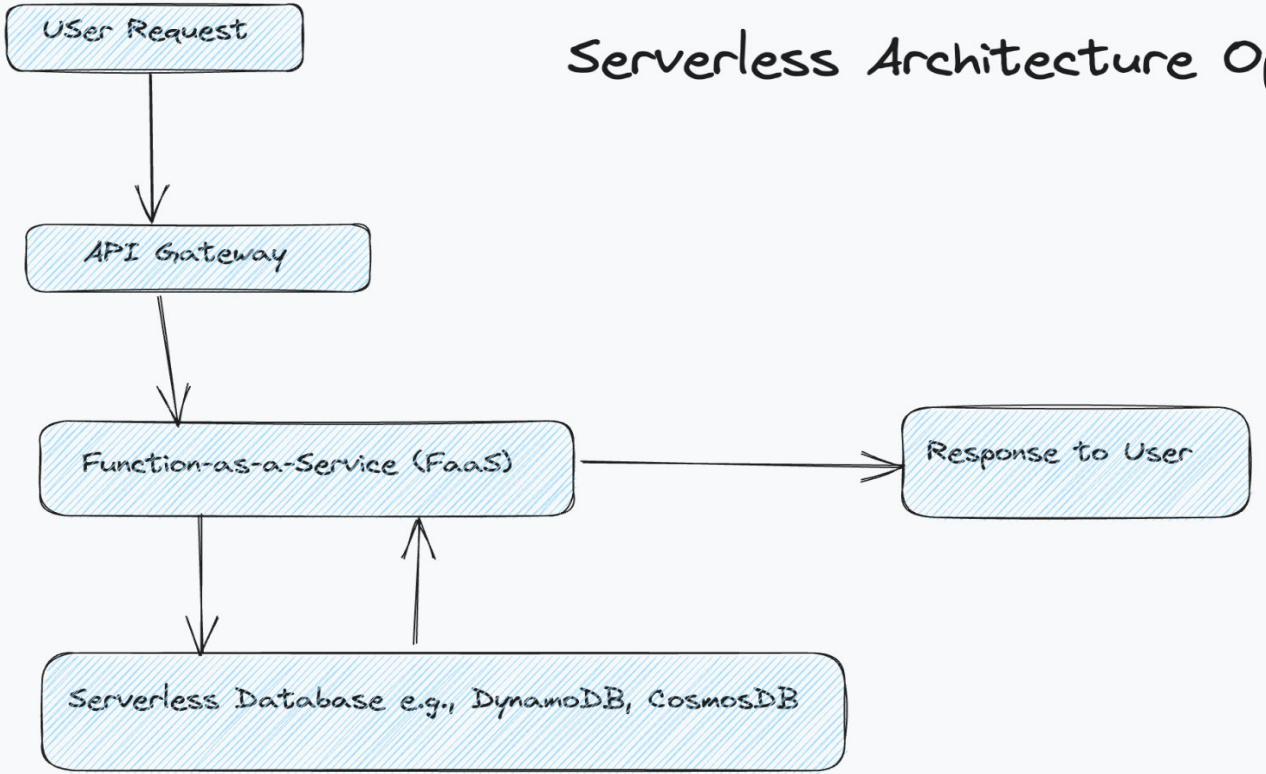
Cost Savings: It can be cheaper to store and manage large amounts of data in a data lake compared to other systems

What is Serverless Architecture?

Serverless architecture is a design pattern where the management of servers and infrastructure is handled by cloud providers. Instead of thinking about server maintenance, scaling, or capacity, you just focus on your code. When you run your code, the cloud provider automatically allocates the resources, and when the code completes, those resources are released.

Example: AWS Lambda

Serverless Architecture Operations



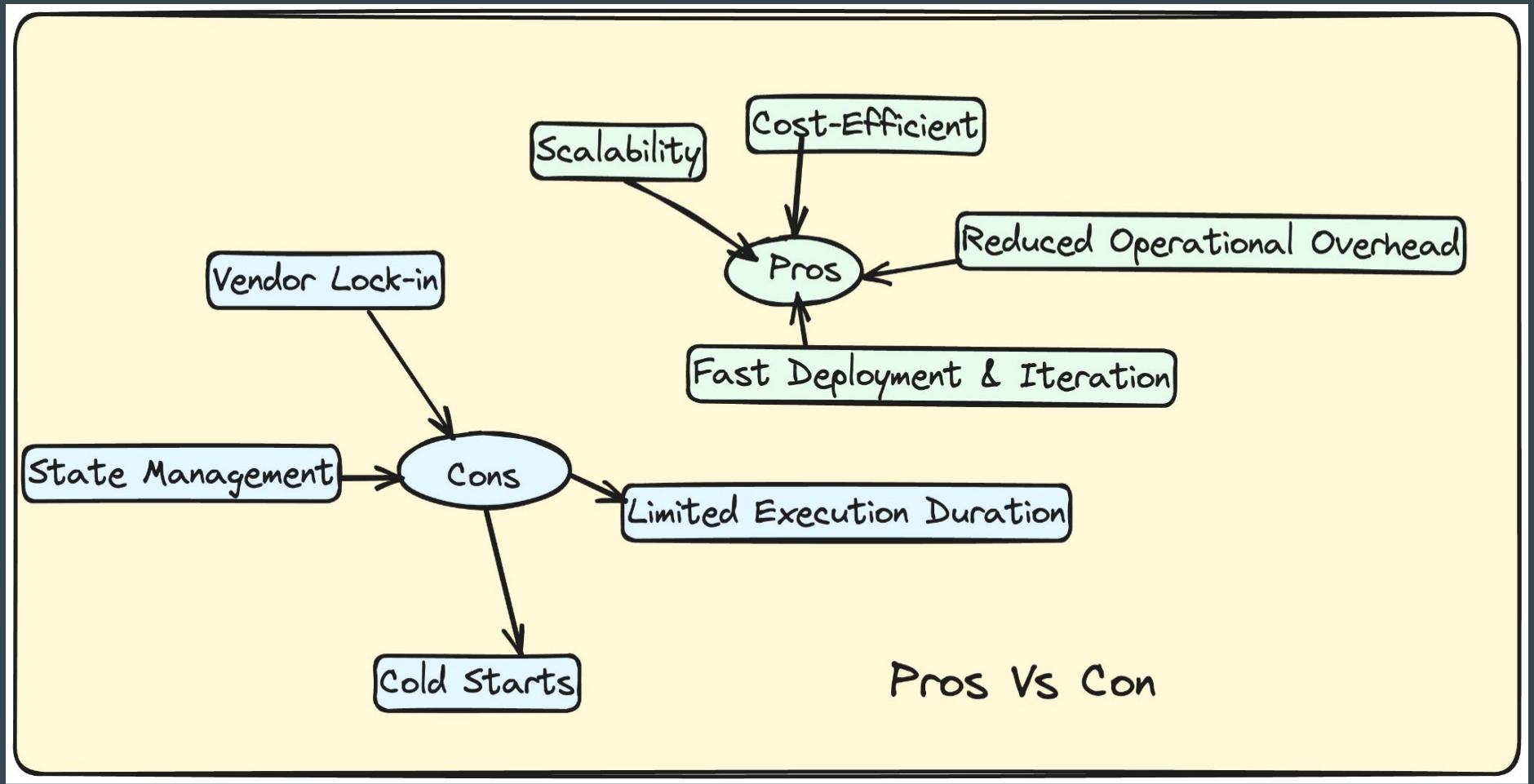
When Serverless

Serverless is an excellent option for projects that want to move quickly without worrying about infrastructure, especially for sporadic or unpredictable workloads.However, it comes with trade-offs.

AWS→ Lambda, Step Functions, Dynamo DB

GCP→ Cloud Functions, Cloud run, etc

Azure → Functions, Logic apps, etc


Scalability
Cost-Efficient
Reduced Operational Overhead
Pros
Fast Deployment & Iteration
Vendor Lock-in
State Management
Cons
Limited Execution Duration
Cold Starts
Pros Vs Con

Pros of Serverless Architecture:

No Server Management: You don't need to provision, scale, or maintain servers.

Cost Effective: You only pay for the compute time your code runs, not for idle server time.

Automatic Scaling: It scales automatically with the number of requests, without manual intervention.

Quick Deployments & Updates: Simplified deployment process often results in faster development cycles.

High Availability: Cloud providers ensure that the infrastructure is fault-tolerant and highly available.

Cons of Serverless Architecture:

Cold Starts: The first time a function runs (or after it hasn't been run for a while), it can have an added delay called a "cold start".

Limited Customization: There might be restrictions on runtime, memory, execution duration, etc.

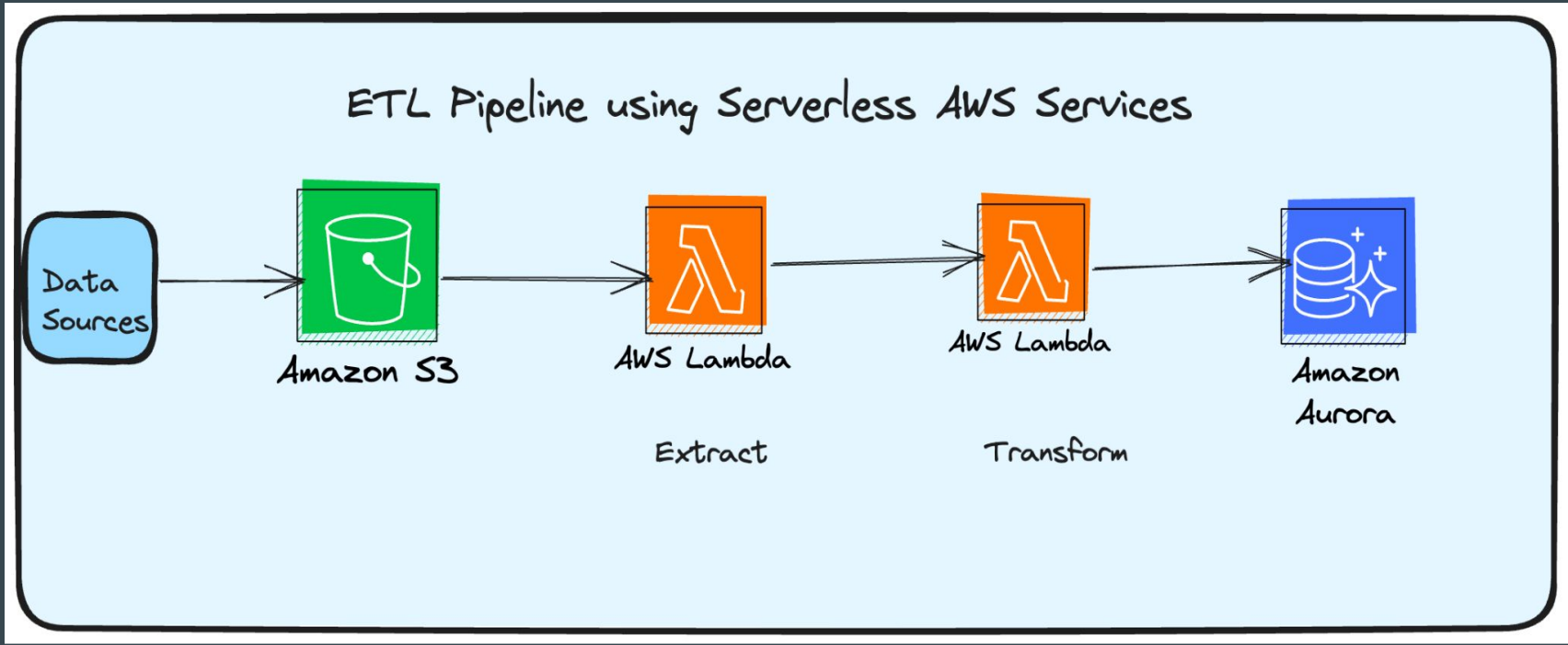
State Management: Serverless functions are stateless, meaning managing and storing state can be challenging.

Debugging & Monitoring: Traditional debugging tools might not work in a serverless environment.

Vendor Lock-in: Relying heavily on a cloud provider's implementation of serverless can make it hard to switch providers later.

Cost : If not monitored properly, a sudden surge in requests might lead to unexpected costs.

Diagram of ETL Pipeline using Serverless AWS Services



Pipeline

Data Sources:

This can be anything from web traffic logs, databases, third-party APIs, to IoT sensors.

Amazon S3:

- Serves as the primary landing zone for raw data.
- It's durable and cost-effective for storing large amounts of unstructured or structured data.
- Data from various sources is ingested into S3.
- Once new data is dumped into S3, it can trigger a Lambda function for further processing.

Pipeline-Continued

Event Trigger:

- Notify when new data arrives.
- AWS Lambda can be set up to automatically execute code in response to notifications from Amazon S3 (like when a new file is saved).

Amazon Lambda (Extract):

- Serverless compute service that can run code in response to certain events without requiring any infrastructure management.
- After being triggered by the new data in S3, this Lambda function will extract the relevant information from the raw data.

Pipeline-Continued

Amazon Lambda (Transform):

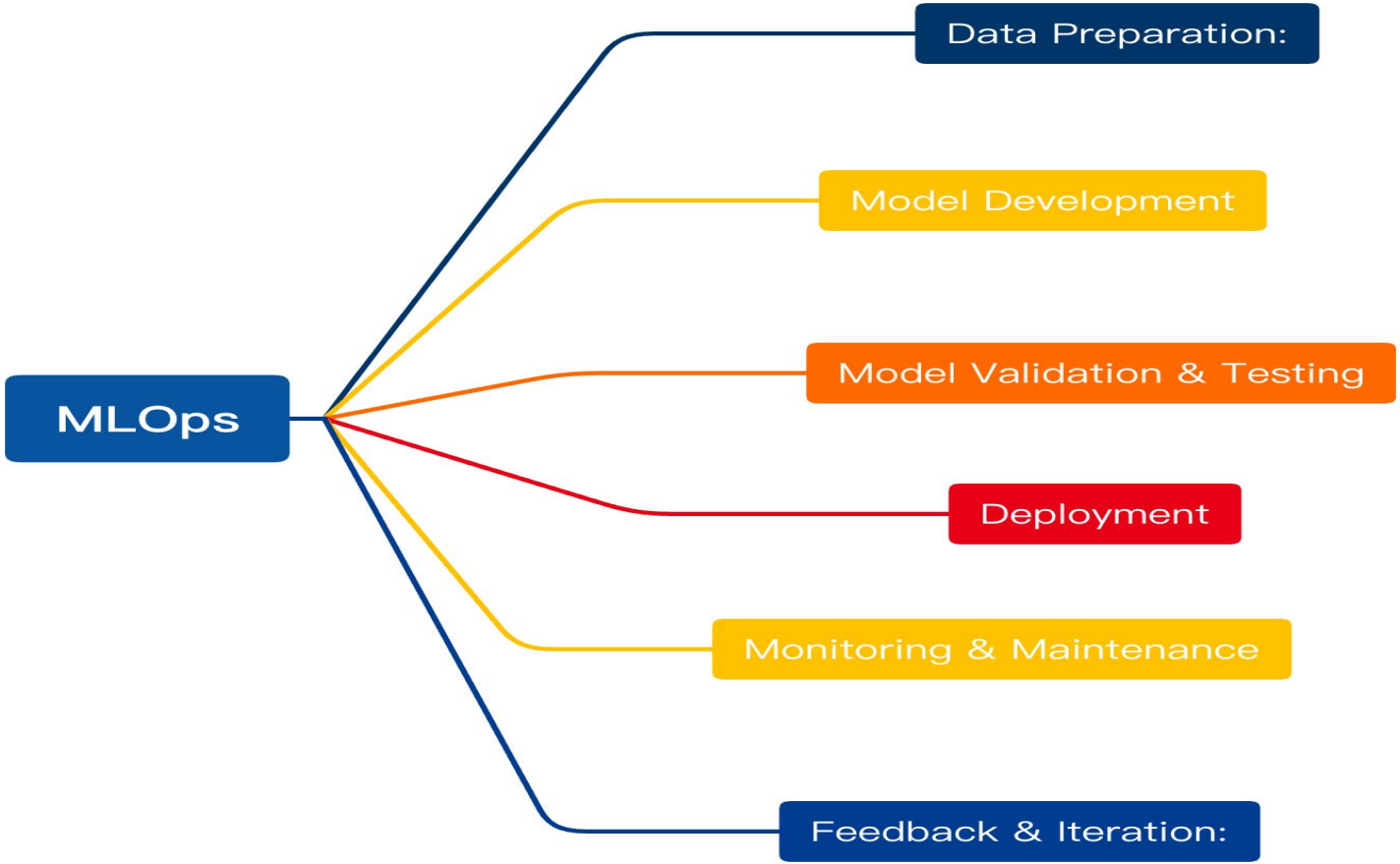
- Another Lambda function in the pipeline that's responsible for the transformation phase.
- Transforms the extracted data (cleaning, enrichment, aggregating, filtering, etc.) into a format suitable for analytics or loading to the destination database.

Amazon RDS/Aurora/Redshift (Load):

- These are managed relational database services from AWS. Redshift, specifically, is designed for OLAP (Online Analytical Processing) and can handle large scale data warehousing.
- The transformed data from the Lambda function is then loaded into one of these databases for analytics or reporting.Here we have Aurora

MLOps

- MLOps is a set of practices and tools that unifies machine learning system development and operations (Ops).
- It aims to automate and streamline the end-to-end machine learning lifecycle, from data preparation and model training to deployment and monitoring, ensuring faster, more reliable, and continuous delivery of ML models to production.


MLOps
Data Preparation:
Model Development
Model Validation & Testing
Deployment
Monitoring & Maintenance
Feedback & Iteration:

Key principles of MLOps include:

Versioning: Everything from raw data, processed datasets, feature engineering scripts, to models should be versioned, ensuring reproducibility.

Automated Testing: Before deploying, models should be tested for their performance, robustness, and fairness.

Continuous Integration and Continuous Delivery (CI/CD) for ML: Automate the building, testing, and deployment processes for ML models, similar to software deployment.

Monitoring and Logging: Continuous monitoring of model performance in production, detecting data drift, model decay, and logging predictions for audit purposes.

Model Governance and Auditability: Ensuring that models meet regulatory requirements, have proper documentation, and can be audited.

Collaboration and Communication: Promote a seamless collaboration environment where data scientists, ML engineers, and operations can work together effectively.

MLOps and how organizations should approach it from a tools and systems perspective:

- **Data Versioning Tools:** Tools like DVC (Data Version Control) help in versioning datasets and ML models similar to how Git versions code.
- **Pipeline Orchestration:** Tools like Apache Airflow or Prefect can be used to automate ML workflows, ensuring that processes like data collection, preprocessing, and model training happen in a structured manner.
- **Experiment Tracking:** Use tools like MLflow or TensorBoard to track and compare different model experiments.
- **Automated Deployment:** Platforms like TFX (TensorFlow Extended) or tools like Seldon or Kubeflow can help in the seamless deployment of ML models.

MLOps and how organizations should approach it from a tools and systems perspective:

- **Model Monitoring:** Monitor models in production using tools like Prometheus, Grafana, or specialized solutions like ModelDB or Fiddler.
- **Containerization and Orchestration:** Use Docker for containerizing ML models to ensure consistent environments. Kubernetes can be used to orchestrate these containers, ensuring scalability and reliability.
- **Infrastructure as Code (IaC):** Use tools like Terraform or AWS CloudFormation to codify infrastructure requirements, making it easier to replicate ML environments.
- **Collaboration Tools:** Platforms like Jupyter or tools like Git promote collaboration among team members, helping them share and review code, data, and models.

MLOps and how organizations should approach it from a tools and systems perspective:

- **Central Model Registry:** Maintain a central repository of models, their versions, and metadata, facilitating rollbacks, audits, and compliance.
- **Feedback Loops:** Set up systems to obtain feedback on model predictions, which can be used for active learning or model retraining.

Conclusion: It's not just about tools but also culture. It requires a cultural shift, where ML development is seen through the lens of software engineering principles, and where collaboration and automation are prioritized. By integrating these tools and systems and fostering the right culture, organizations can achieve faster, more reliable, and more accountable ML deployments.

Version Control

Experiment Tracking

Continuous Integration & Continuous Deployment (CI/CD)

Model Serving & Deployment

Infrastructure Management

Pipeline Building (Orchestration & Workflow Management)

Interactive Applications (for Rapid Prototyping & Model Visualization):


Flask


Streamlit

Model Monitoring

Section 2: Database & Python ETL

Selection of Database: SQLite

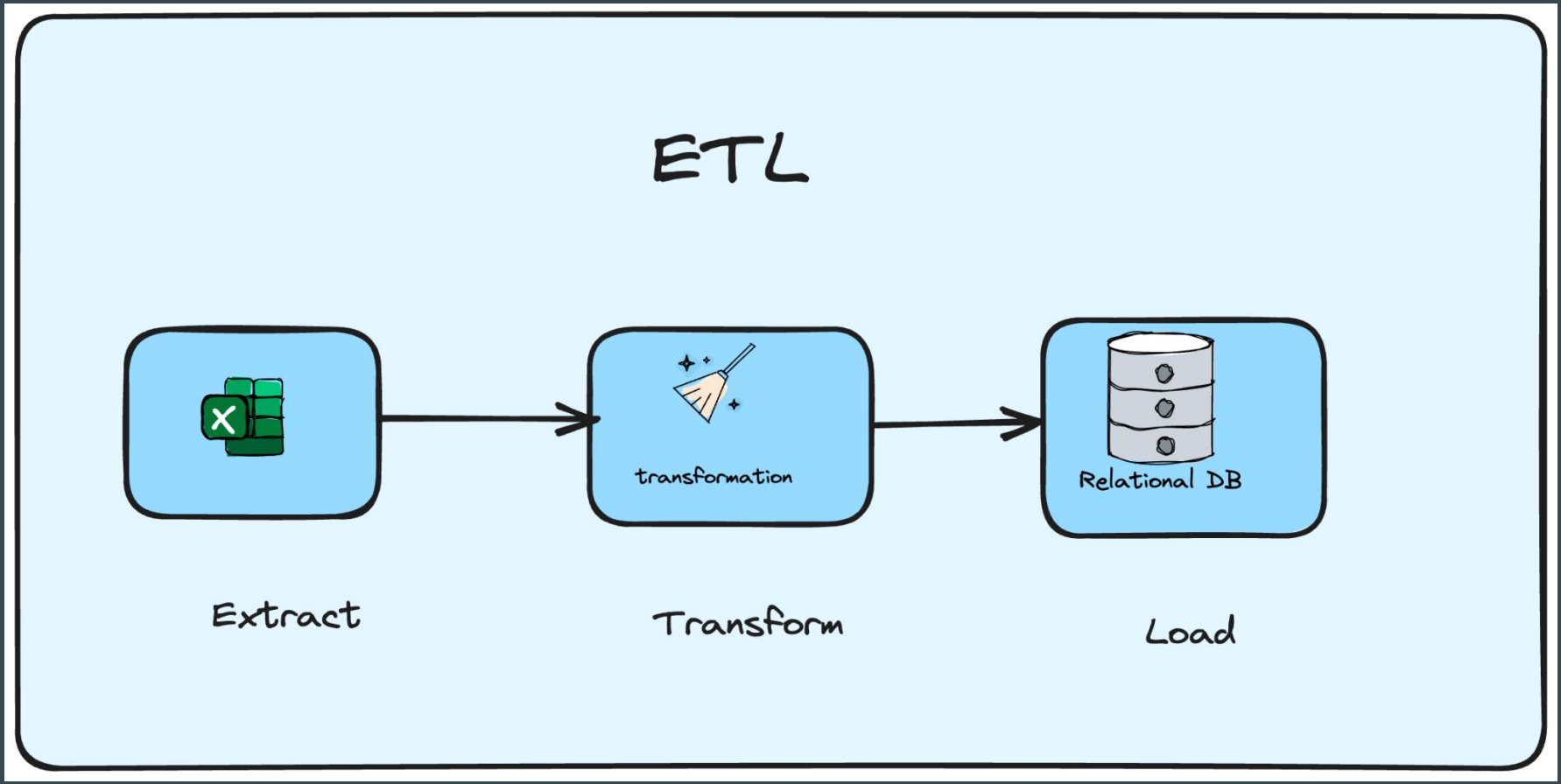
SQLite is a lightweight, serverless, and self-contained SQL database engine. It's great for local development and prototyping.

ETL:

Extract: Load the data from the provided Excel file.

Transform: Create one or two new fields and apply transformations where needed.

Load: Insert the transformed data into the SQLite database.


ETL
transformation
Relational DB
Extract
Transform
Load

Examine the Dataset

The dataset contains the following columns:

id: A unique identifier for each entry.

first_name: First name of the person.

last_name: Last name of the person.

email: Email address of the person.

gender: Gender of the person.

ip_address: IP address associated with the person.

A	B	C	D	E	F
id	first_name	last_name	email	gender	ip_address
1	Margaretta	Laughtisse	mlaughtisse0@mediafire.com	Genderfluid	34.148.232.131
2	Vally	Garment	vgarment1@wisc.edu	Bigender	15.158.123.36
3	Tessa	Curee	tcuree2@php.net	Bigender	132.209.143.225
4	Arman	Heineking	aheineking3@tuttocitta.it	Male	157.110.61.233
5	Roselia	Trustie	rtrustie4@ft.com	Non-binary	49.55.218.81
6	Roxie	Springett	rspringett5@deviantart.com	Male	51.206.104.138
7	Gabi	Kernell	gkernell6@hugedomains.com	Female	223.30.27.146
8	Dino	Kentwell	dkentwell7@com.com	Agender	107.244.52.181
9	Petronilla	Jandel	pjandel8@amazon.co.uk	Female	187.54.208.203
10	Courtnay	Zecchinelli	czecchinelli9@cam.ac.uk	Genderfluid	80.96.245.191
11	Sunny	Kennermann	skennermanna@quantcast.com	Genderqueer	211.13.246.106
12	Dayle	McCrachen	dmccrachenb@booking.com	Genderqueer	159.232.55.236
13	Cassie	Perschke	cperschkec@goo.gl	Genderqueer	193.62.46.4
14	Roshelle	Peskin	rpeskind@patch.com	Genderqueer	108.180.147.192
15	Carlie	Simonnot	csimonnote@surveymonkey.com	Female	220.154.167.3
16	Gan	Siuda	gsiudaf@ucoz.ru	Agender	118.117.172.157
17	Klarika	Filimore	kfilimoreg@creativecommons.org	Agender	16.111.119.168
18	Lennie	Bilbrook	lbilbrookh@hatena.ne.jp	Bigender	162.30.71.206
19	August	Cristoforo	acristoforoi@goo.ne.jp	Genderfluid	30.231.88.242
20	Avrit	Milburne	amilburnej@yahoo.com	Bigender	186.73.151.205
21	Freeland	Sperling	fsperlingk@a8.net	Genderqueer	107.242.151.123
22	Brice	Elintune	belintunel@mlb.com	Non-binary	217.244.45.24

Basic Stats:

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   id           1000 non-null   int64 
 1   first_name   1000 non-null   object
 2   last_name    1000 non-null   object
 3   email        1000 non-null   object
 4   gender       1000 non-null   object
 5   ip_address   1000 non-null   object
dtypes: int64(1), object(5)
memory usage: 47.0+ KB
None
```

```
print(data.describe())
```

```
                id
count  1000.000000
mean    500.500000
std     288.819436
min       1.000000
25%     250.750000
50%     500.500000
75%     750.250000
max    1000.000000
```

Walk through the Code

Extract: Read the data from the provided Excel file using pandas

```
[1]  import pandas as pd

     # Load the dataset from the provided Excel file
     data = pd.read_excel("/content/sample_data/SRDataEngineerChallenge_DATASET.xlsx"

     # Display the first few rows to get an understanding of the dataset's structure
     data.head()
```

Extract

Display the dataframe and check few records

	id	first_name	last_name	email	gender	ip_address
0	1	Margaretta	Laughtisse	mlaughtisse0@mediafire.com	Genderfluid	34.148.232.131
1	2	Vally	Garment	vgarment1@wisc.edu	Bigender	15.158.123.36
2	3	Tessa	Curee	tcuree2@php.net	Bigender	132.209.143.225
3	4	Arman	Heineking	aheineking3@tuttocitta.it	Male	157.110.61.233
4	5	Roselia	Trustie	rtrustie4@ft.com	Non-binary	49.55.218.81

Transformations Applied

Full Name: We can combine the first_name and last_name columns to create a full_name column.

data['full_name'] = data['first_name'] + " " + data['last_name']

Domain of Email: We can extract the domain from the email column to understand which email service providers are most popular.

data['email_domain'] = data['email'].str.split('@').str[1]

Transformations

```
[2] # Transformation 1: Create 'full_name' column by combining 'first_name' and 'last_name'
    data['full_name'] = data['first_name'] + " " + data['last_name']

    # Transformation 2: Extract the domain from the 'email' column
    data['email_domain'] = data['email'].str.split('@').str[1]

    # Display the first few rows after transformations
    data.head()
```

Transformations

Full Name and Email Domain are added

	id	first_name	last_name	email	gender	ip_address	full_name	email_domain
0	1	Margaretta	Laughtisse	mlaughtisse0@mediafire.com	Genderfluid	34.148.232.131	Margaretta Laughtisse	mediafire.com
1	2	Vally	Garment	vgarment1@wisc.edu	Bigender	15.158.123.36	Vally Garment	wisc.edu
2	3	Tessa	Curee	tcuree2@php.net	Bigender	132.209.143.225	Tessa Curee	php.net
3	4	Arman	Heineking	aheineking3@tuttocitta.it	Male	157.110.61.233	Arman Heineking	tuttocitta.it
4	5	Roselia	Trustie	rtrustie4@ft.com	Non-binary	49.55.218.81	Roselia Trustie	ft.com

Load

Now load the transformed data into SQLite database table.

```
import sqlite3

# Create an SQLite database in memory
conn = sqlite3.connect(':memory:')
cursor = conn.cursor()

# Load the transformed data into the SQLite database
data.to_sql('users', conn, if_exists='replace', index=False)

# Verify the data has been loaded by fetching the first few rows from the 'users' table
query = "SELECT * FROM users LIMIT 5"
pd.read_sql(query, conn)
```

Load

Query the table and see the output

	id	first_name	last_name	email	gender	ip_address	full_name	email_domain
0	1	Margaretta	Laughtisse	mlaughtisse0@mediafire.com	Genderfluid	34.148.232.131	Margaretta Laughtisse	mediafire.com
1	2	Vally	Garment	vgarment1@wisc.edu	Bigender	15.158.123.36	Vally Garment	wisc.edu
2	3	Tessa	Curee	tcuree2@php.net	Bigender	132.209.143.225	Tessa Curee	php.net
3	4	Arman	Heineking	aheineking3@tuttocitta.it	Male	157.110.61.233	Arman Heineking	tuttocitta.it
4	5	Roselia	Trustie	rtrustie4@ft.com	Non-binary	49.55.218.81	Roselia Trustie	ft.com

Load Data

SQLite3 Import:

The sqlite3 library is imported, which allows for interaction with SQLite databases in Python.

Database Creation:

An SQLite database is created in memory. This means the database is temporary and will be discarded once the connection is closed.

A cursor object (cursor) is created, which will be used to execute SQL commands.

Data Loading:

The DataFrame data is loaded into the SQLite database.

A new table named users is created in the database to hold this data.

If a table named users already exists, it will be replaced with the new data. This is due to the if_exists='replace' parameter.

The index=False parameter ensures that the DataFrame's index is not saved as a separate column in the database.

Data Verification:

A SQL query is constructed to fetch the first five rows from the users table.

The pd.read_sql function is used to execute this query, and the result is returned as a DataFrame.

Replicate

1.Clone the Repository:

First, clone the GitHub repository containing the ETL script and associated files.

```
git clone https://github.com/esenthil2018/Data_Engineering_2.git
cd Data_Engineering_2
```

Replicate

2.Docker File:

```
# Use an official Python runtime as the base image

FROM python:3.9-slim

# Set the working directory in the container to /app

WORKDIR /app

# Copy the current directory contents into the container at /app

COPY . /app

# Install required packages

RUN pip install --no-cache-dir -r requirements.txt

# Run the ETL script when the container is started

CMD ["python", "etl_script.py"]
```

Replicate

3.requirements.txt

pandas

sqlite3

openpyxl

Replicate

Build and run:

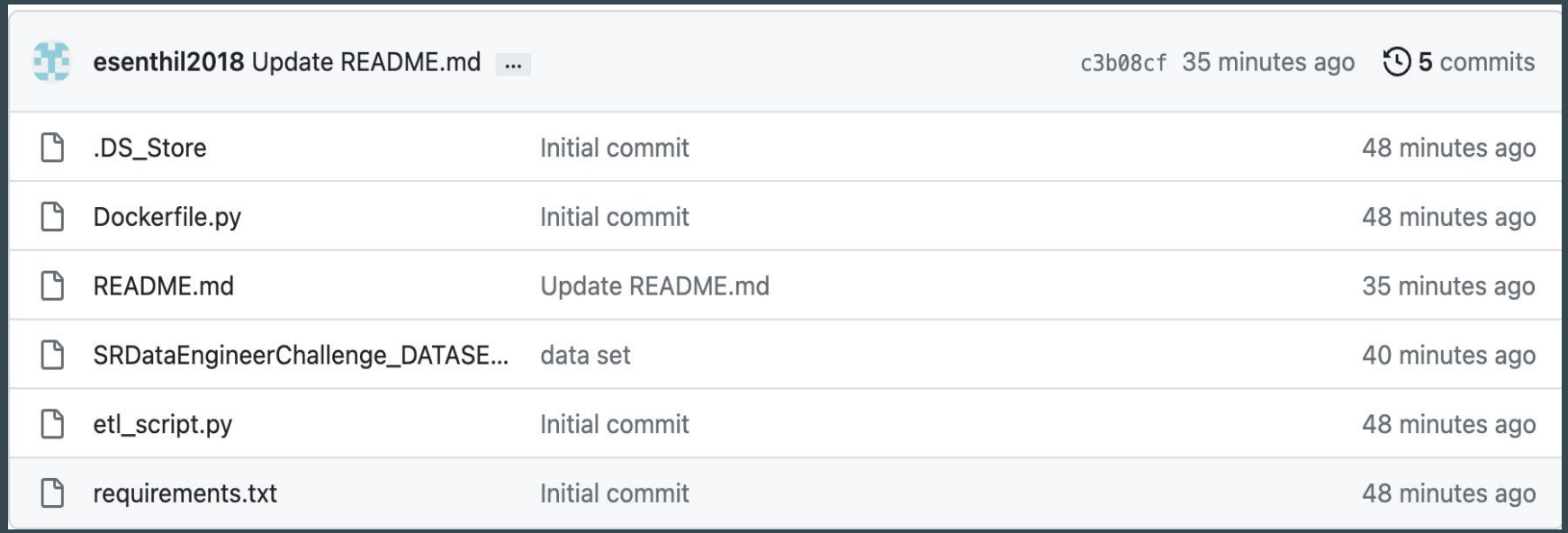
docker build -t etl-image .

docker run etl-image

Once the container is running, the ETL script (etl_script.py) will automatically start processing the data. The script will extract data from the provided dataset, perform transformations, and load it into an SQLite database in memory.

Github Repository Link

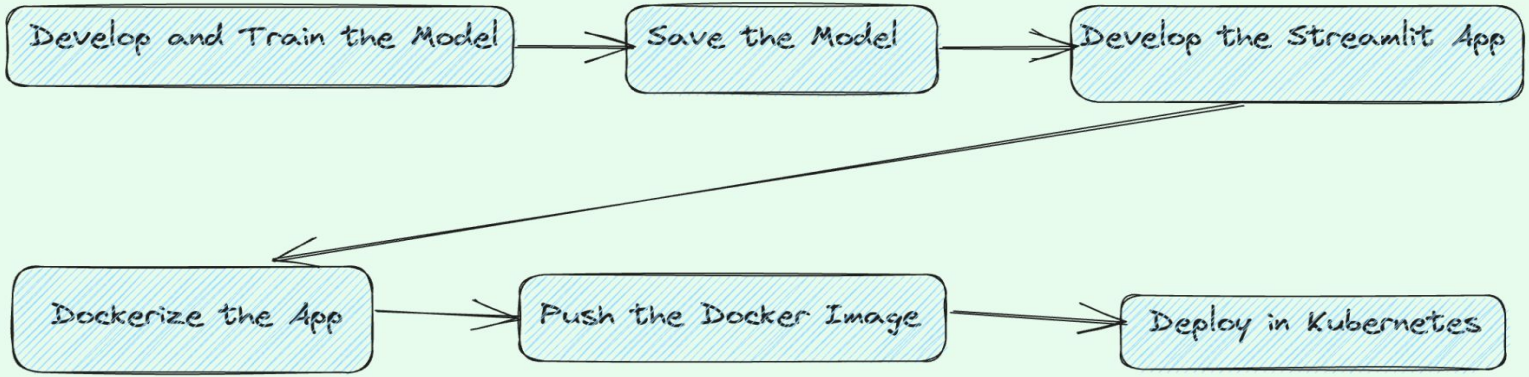
https://github.com/esenthil2018/Data_Engineering_2



esenthil2018 Update README.md ...	`c3b08cf` 35 minutes ago	**5** commits
.DS_Store	Initial commit	48 minutes ago
Dockerfile.py	Initial commit	48 minutes ago
README.md	Update README.md	35 minutes ago
SRDataEngineerChallenge_DATASE...	data set	40 minutes ago
etl_script.py	Initial commit	48 minutes ago
requirements.txt	Initial commit	48 minutes ago

Section:3-ML API

On a cloud (AWS, Azure, GCP) or local environment (docker, kubernetes), deploy an ML model (ex. MNIST: https://paperswithcode.com/dataset/mnist, Fashion MNIST: https://github.com/zalandoresearch/fashion-mnist, or other) as an API endpoint. Please provide reproducible code with documentation for both deployment and usage.



Model Deployment Process

Environment

Step 1: Set Up the Environment

gcloud CLI tool installed: https://cloud.google.com/sdk/gcloud

kubectl installed: https://kubernetes.io/docs/tasks/tools/

A Google Cloud account and a GKE cluster running.

Libraries

 # For deep learning model creation and training

tensorflow

 # For web application framework

streamlit

 # For image handling in the Streamlit app

PILLOW

 # For numerical operations (used indirectly by TensorFlow and Streamlit)

numpy

 # For Google Cloud SDK operations

google-cloud-sdk

Step:1-Train and Save the MNIST Model

- Load the MNIST dataset.
- Pre-process and normalize the data.
- Construct a neural network with specific layers and activation functions.
- Compile the model specifying the optimizer, loss function, and evaluation metrics.
- Train the model using the training dataset.
- Save the trained model to a .h5 file for future use.

Step:1-Train and Save the MNIST Model

```
import tensorflow as tf

# Load the MNIST dataset
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize the pixel values (from 0 to 255) to the 0-1 range
x_train, x_test = x_train / 255.0, x_test / 255.0

# Build the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=5)

# Save the model
model.save('mnist_model.h5')
```

Step:2-Create the Streamlit App

- Load the pre-trained MNIST model.
- Define a function to process uploaded images and make predictions using the model.
- Design the Streamlit interface to allow image uploads.
- Display the uploaded image and show the prediction results.

Step:2-Create the Streamlit App

```
import streamlit as st
import tensorflow as tf
from PIL import Image, ImageOps
import numpy as np

# Load pretrained model
model = tf.keras.models.load_model('mnist_model.h5')

def predict_image(img):
    img = ImageOps.grayscale(img)
    img = img.resize((28, 28))
    img_array = np.array(img) / 255.0
    img_array = img_array.reshape(1, 28, 28)
    predictions = model.predict(img_array)
    predicted_class = np.argmax(predictions, axis=1)[0]
    return predicted_class

st.title("MNIST Digit Classifier")
uploaded_file = st.file_uploader("Upload an image of a handwritten digit", type="jpg")

if uploaded_file is not None:
    image = Image.open(uploaded_file)
    st.image(image, caption='Uploaded Image.', use_column_width=True)
    st.write("Predicting...")
    label = predict_image(image)
    st.write(f'Predicted label: {label}')
```

Step:3-Dockerize the App

- Create a Dockerfile specifying the base image, working directory, dependencies, and the command to run the Streamlit app.
- Build the Docker image.
- Push the Docker image to a container registry (e.g., Google Container Registry).

Step:3-Dockerize the App

```
FROM python:3.8-slim

WORKDIR /app

COPY . /app

RUN pip install streamlit tensorflow numpy pillow

CMD ["streamlit", "run", "app.py"]
```

Step3: Create the Image and Push to the Registry

```
# Build the docker image

docker build -t gcr.io/mlops-project-8934/mnist-streamlit:v1 .

gcloud auth configure-docker

#Push the docker image to the GCP registry

docker push gcr.io/mlops-project-8934/mnist-streamlit:v1
```

Step4: Deploy on GKE(Google Kubernetes Engine)

Define a Kubernetes Deployment configuration to specify how the app should be deployed on GKE.

Define a Kubernetes Service configuration to expose the app to the internet.

Use kubectl commands to apply these configurations and deploy the app on GKE.

Retrieve the external IP to access the deployed Streamlit app.

GKE Deployment Steps

1. Set Up GKE Cluster:
   - Install the Google Cloud SDK.
   - Authenticate your command line to GCP

```
gcloud auth login
```

2. Set your GCP project:

```
gcloud config set project YOUR_PROJECT_ID
```

3. Create a GKE cluster:

```
gcloud container clusters create YOUR_CLUSTER_NAME --zone YOUR_ZONE
--num-nodes=1 --machine-type=e2-small
```

GKE Deployment Steps-Continued

4.Configure kubectl to use the newly created cluster:

```
gcloud container clusters get-credentials YOUR_CLUSTER_NAME --zone YOUR_ZONE
```

5.Push Docker Image to Google Container Registry (GCR):

```
docker tag YOUR_DOCKER_IMAGE gcr.io/YOUR_PROJECT_ID/YOUR_IMAGE_NAME:YOUR_TAG
```

6.Push the Docker image:

```
docker push gcr.io/YOUR_PROJECT_ID/YOUR_IMAGE_NAME:YOUR_TAG
```

GKE-Deployment Steps-Continued

Deployment.yaml:

```
kubectl apply -f deployment.yaml
```

Apply the Service:

```
kubectl apply -f service.yaml
```

View Running Pods:

```
kubectl get pods
```

View active services:

```
kubectl get svc
```

GKE Deployment Steps-Continued

Describe a specific pod

```
kubectl describe pod POD_NAME
```

View logs for a specific pod:

```
kubectl logs POD_NAME
```

Delete a service or deployment:

```
kubectl delete -f FILE_NAME.yaml
```

Accessing Your App:

```
kubectl get svc SERVICE_NAME
```

Deployment Configuration (deployment.yaml):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mnist-streamlit
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mnist-streamlit
  template:
    metadata:
      labels:
        app: mnist-streamlit
    spec:
      containers:
      - name: mnist-streamlit
        image: gcr.io/mlops-project-8934/mnist-streamlit:v1
```

Service Configuration (service.yaml):

Untitle

```
apiVersion: v1
kind: Service
metadata:
  name: mnist-streamlit-service
spec:
  selector:
    app: mnist-streamlit
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8501
  type: LoadBalancer
```

Deployment in GKE

Using kubectl:

Apply the deployment file, Service File and get the endpoint

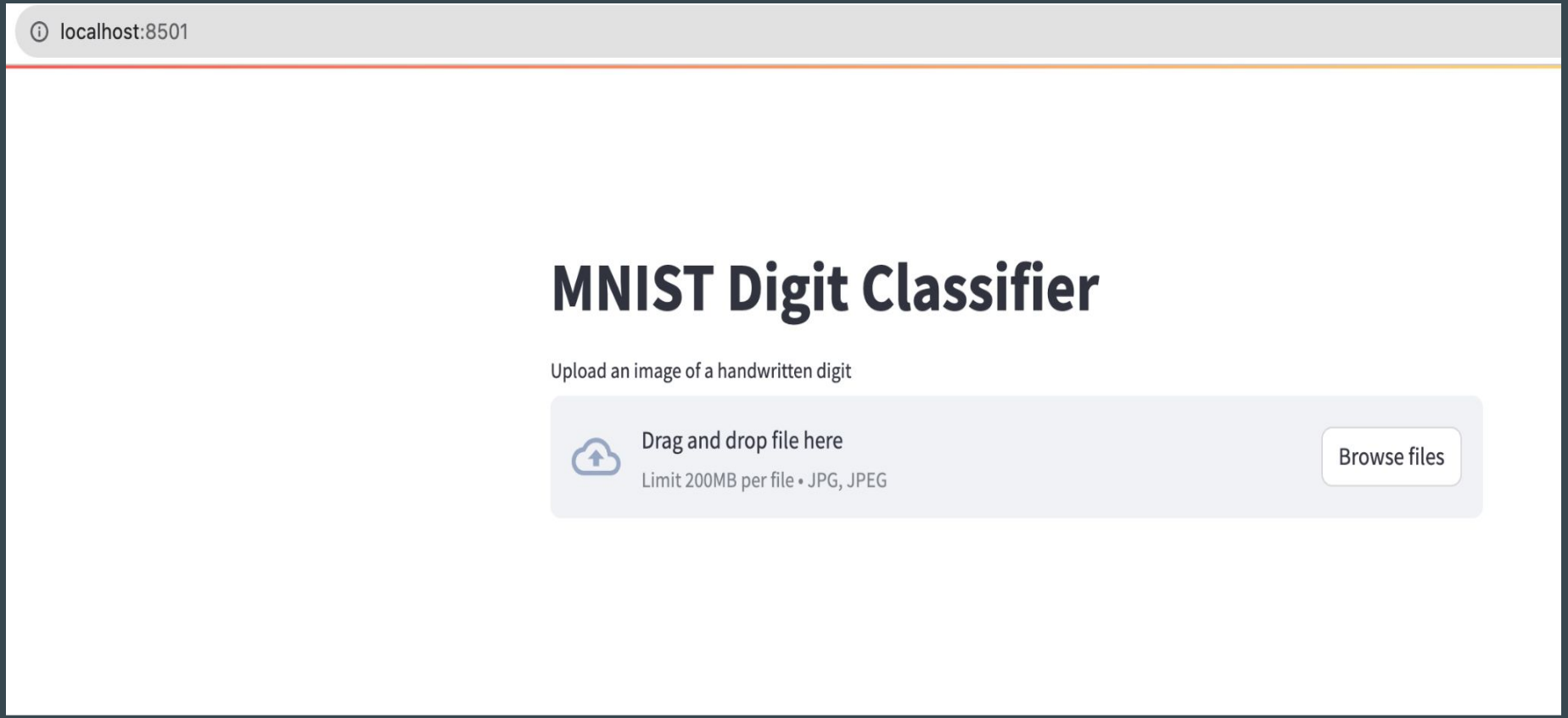
```
kubectl apply -f deployment.yaml
```

```
kubectl apply -f service.yaml
```

```
kubectl get svc mnist-streamlit-service
```

Step5: Share and Replicate

- Bundle all files, scripts, and configurations into a GitHub repository.
- Share the repository link and instructions with colleagues or others to replicate the deployment process.


localhost:8501
MNIST Digit Classifier
Upload an image of a handwritten digit
Drag and drop file here
Limit 200MB per file • JPG, JPEG
Browse files

MNIST Digit Classifier

Upload an image of a handwritten digit

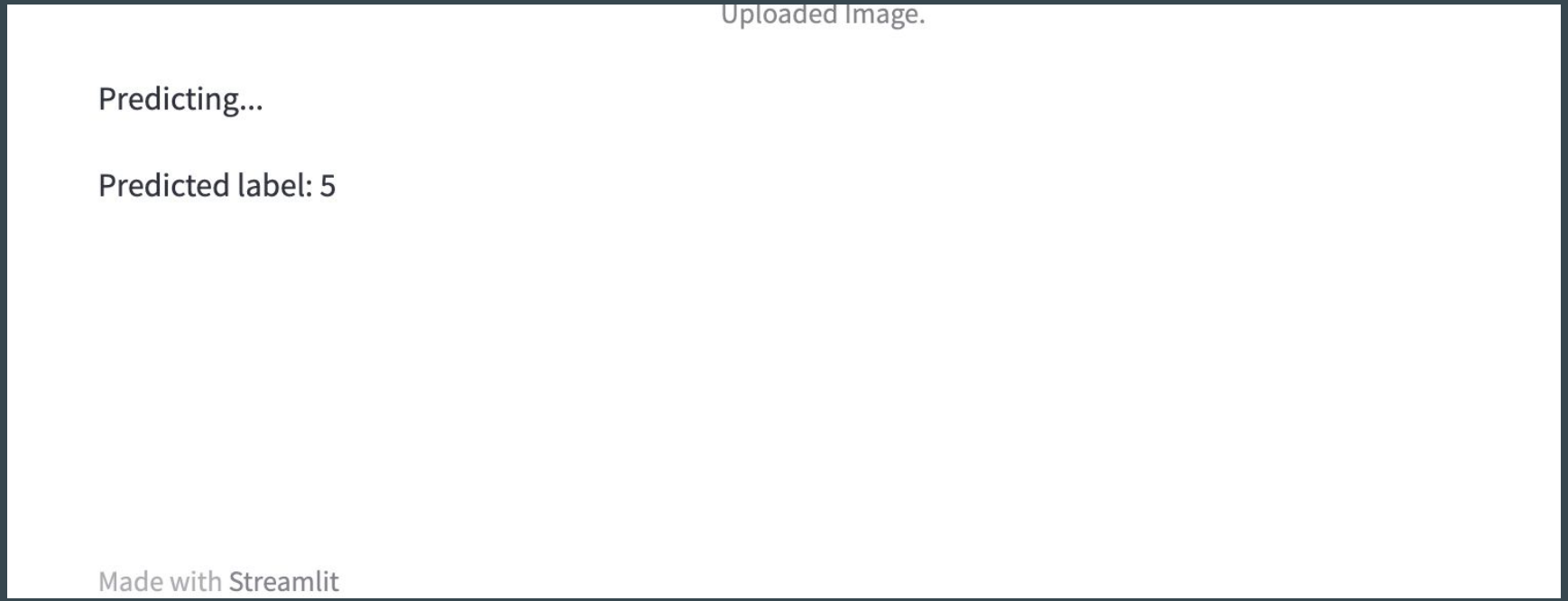
Drag and drop file here

Limit 200MB per file • JPG, JPEG

Browse files

mnist_dream.jpg 104.9KB

Streamlit App



Github Files-Explanation

train_model.py:

The script that defines, trains, and saves the MNIST model.

app.py:

The Streamlit app script. This will load the trained model, allow users to upload images, and display model predictions.

mnist_model.h5 (optional):

The trained model file. This can be large. If you include training instructions, you might decide to leave this out and have users train their own model.

Dockerfile:

Instructions for building your Docker container, which includes the Streamlit app and all its dependencies.

Github Files-Explanation

requirements.txt:

List of Python libraries required for the project.

deployment.yaml:

Kubernetes configuration for deploying your app on Google Kubernetes Engine (GKE).

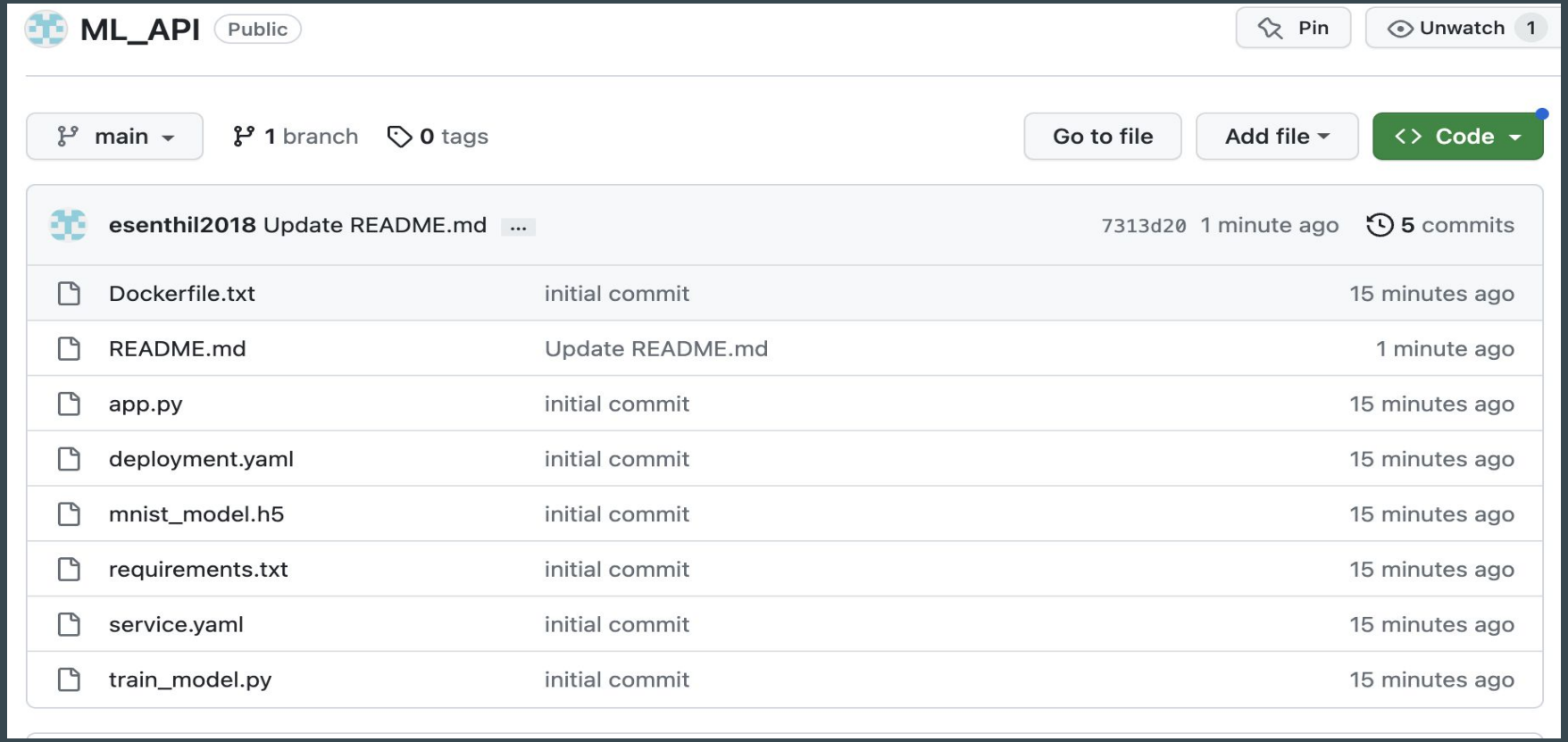
service.yaml:

Kubernetes service configuration to expose your deployed app to the internet.

.gitignore:

A file to specify which files or directories should be ignored by Git.

https://github.com/esenthil2018/ML_API



Replicate:1

Clone the Repository:

git clone https://github.com/esenthil2018/ML_API.git

cd ML_API

Set Up a Virtual Environment:

python -m venv venv

source venv/bin/activate # On Windows, use`venv\Scripts\activate`

Install the Required Packages:

pip install -r requirements.txt

Few of My Articles:

Medium:https://medium.com/@esenthil

MLOps End-To-End Machine Learning Pipeline-CICD

https://medium.com/analytics-vidhya/mlops-end-to-end-machine-learning-pipeline-cicd-1a7907698a8e?sk=0e776f6c255989997ee8ff966067b70f

LineaPy Data Science Workflow In Just Two Lines: MLOps Made Easy

https://towardsdatascience.com/lineapy-data-science-workflow-in-just-two-lines-mlops-made-easy-679f36ac63bd?sk=78a8fa6cf59180eb25177f64ee87d50e

MLOps: Mastering Machine Learning Deployment: An Intro to Docker, Kubernetes, Helm, and Modern Web Frameworks-End To End Project

https://medium.com/gitconnected/mlops-mastering-machine-learning-deployment-an-intro-to-docker-kubernetes-helm-and-modern-web-b14dd140a9bf

Conclusion

- I'd like to sincerely thank the Deloitte team for providing me with this invaluable opportunity. It's been both challenging and rewarding to delve into this case study.
- The process reinforced my desire to be a part of Deloitte. I am eager to contribute, learn, and grow with such a dynamic team.
- I'm open to any questions and feedback. Looking forward to our continued discussions!

Thank You!!