Project Title - Wafer Fault Detection

Artificial Intelligence to develop eco-friendly, sustainable, long life products. Leverage the technological advancements to develop 21st century products.

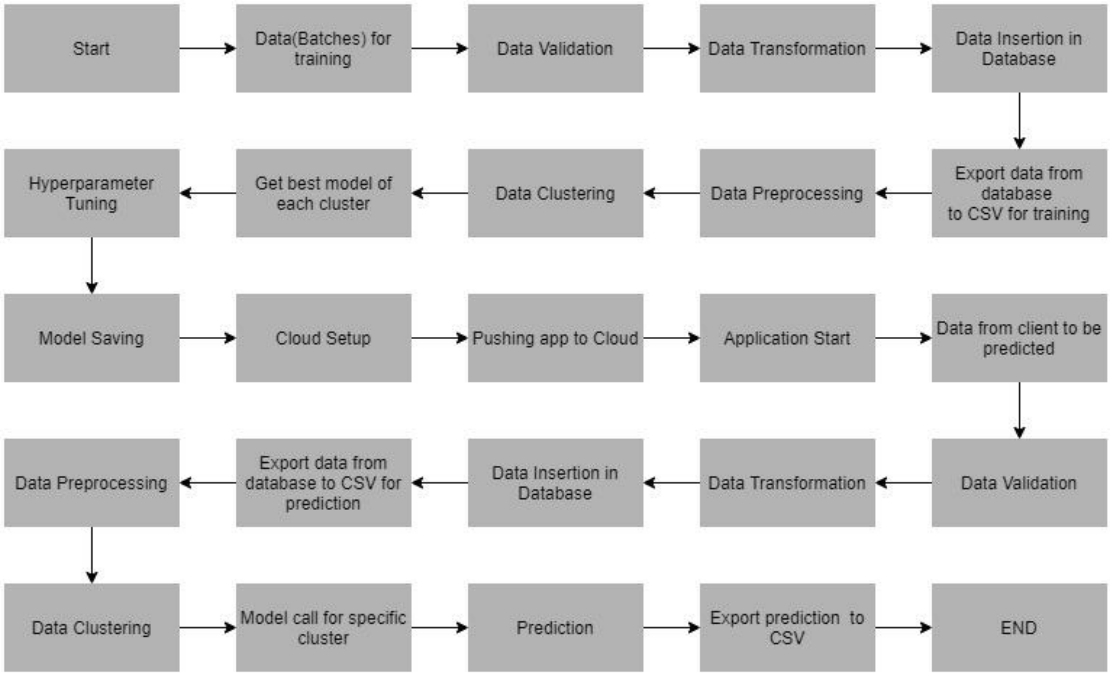
Wafer Fault Detection

INDEX

Business Statement

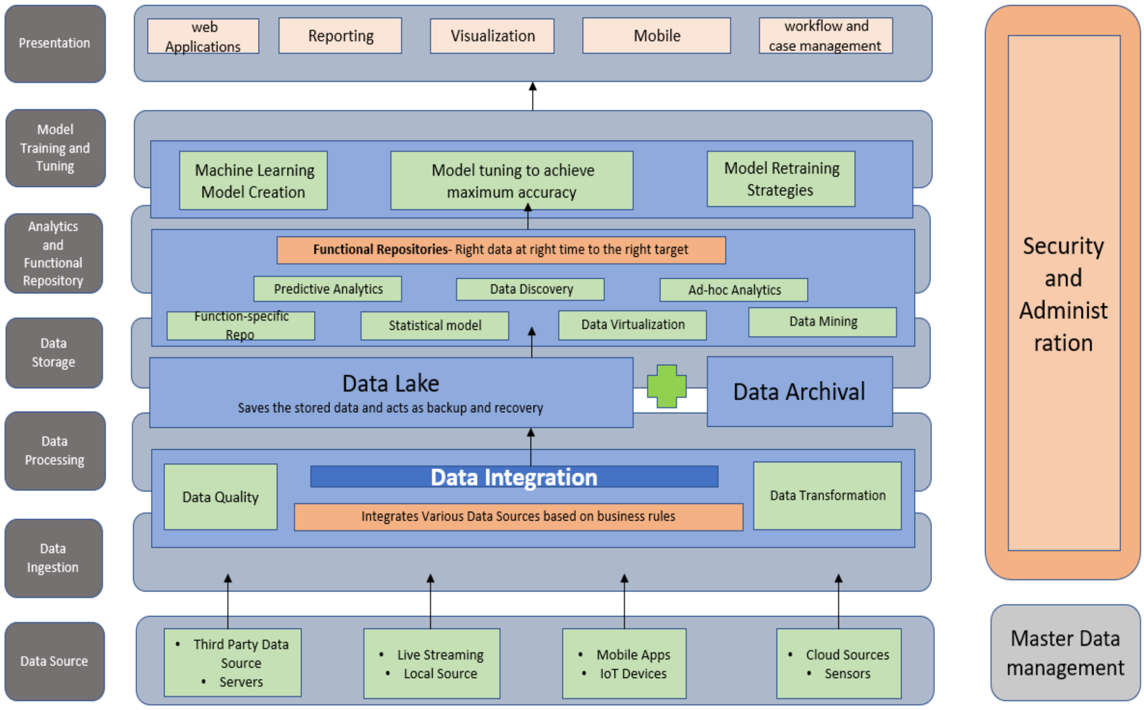
Build a classification methodology to predict the quality of wafer sensors based on the given training data.

Project Architecture

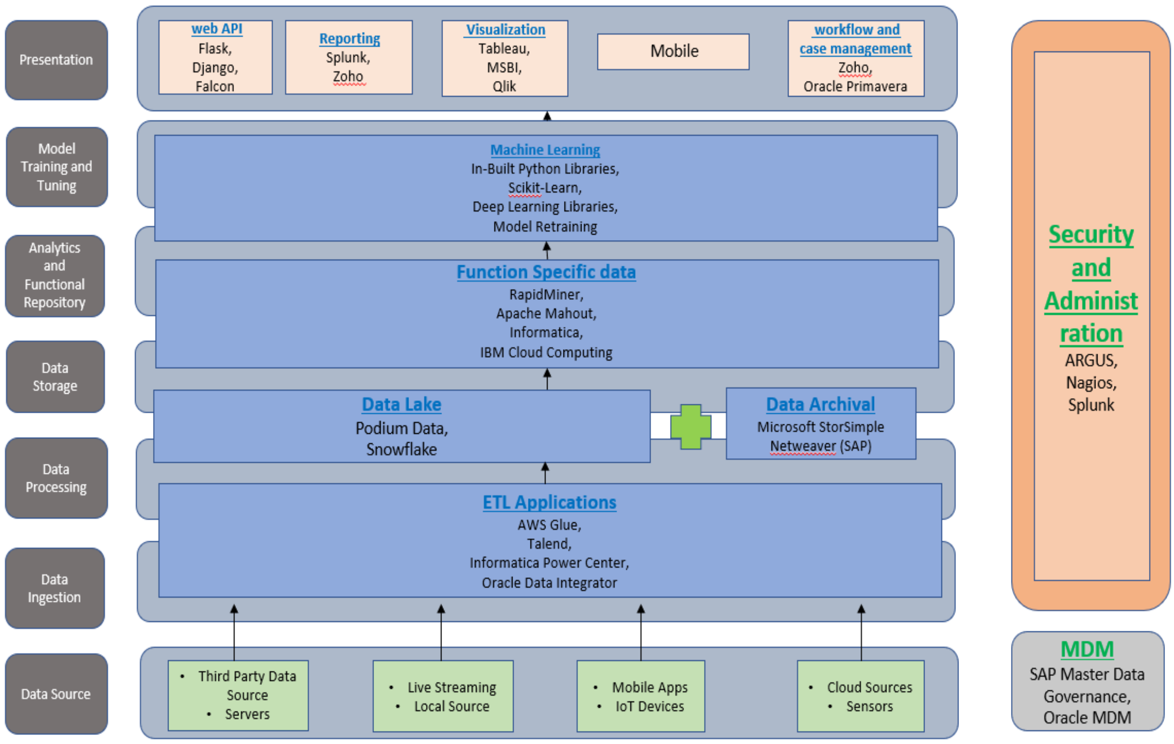


Continued…

Functional Architecture



Technical Architecture



Data Description

The client will send the data in multiple sets of files in batches at a specified location. Data will contain Wafer names and 590 columns of different sensor values for each wafer. The last column will have the “Good/Bad” value for each wafer.

“Good/Bad” column will have two unique values +1 and -1

“+1”: represents bad wafer

“-1“: represents good wafer

Apart from training files, we also require a “schema” file from the client, which contains all the relevant information about the training files such as:

1. Name of the files
2. Length of date value in file name
3. Length of time value in file name
4. Number of columns
5. Name of the columns with datatype

Data Ingestion Validation

In this step, we perform different sets of validation on the given set of training files.

1. Name Validation: We validate the name of the files based on the given name in the schema file. We have created a regex pattern as per the name specified in schema file. After validating the pattern in the name, we check for the length of data in the file name as well as the length of time in the file name. if all the values are as per the requirement, we move such files to “Good_Data” directory else move such files to “Bad_Data”
2. Number of Columns: We validate the number of columns present in the files, and if it doesn’t match with the value given in the schema file, then the file is moved to “Bad_Data”
3. Name of Columns: The name of the columns is validated and should be the same as given in the schema file. If not, then the file is moved to “Bad_Data”
4. The datatype of columns: The datatype of columns is given in the schema file. This is validated when we insert the files into database. If the datatype is wrong, then move the file to “Bad_Data”.
5. Null Values in Column: If any of the columns in a file have all the values as NULL or missing, we discard such a file and move it to “Bad_Data”

Training Data Insertion

1. Database Creation & Connection: create a database named as “waferdb”. If the database is already created, open the connection to the database.
2. Collection Creation in Database: Collection with name – “waferCollection” create in database “waferdb” and insert all the data files from “Good_Data”

Model Training

1. Data Export from DB: The data is stored in MongoDB in database “waferdb” in collection “waferCollection”, export this collection in CSV file for model training
2. Data Preprocessing:
    a. Check for NULL Value: If null value(s) present, impute the null values using KNN imputer.
    b. Columns with Zero Standard Deviation: Identify such columns and drop them as these columns do not contribute in model training.
3. Clustering: KMeans algorithm is used to create clusters in the preprocessed data. The optimum number of clusters is selected by plotting the elbow plot, and for the dynamic selection of the number of clusters, we are using “KneeLocator” function. The idea behind clustering is to implement different algorithms. To train the data in different clusters, the KMeans model is trained over preprocessed data and the model is saved for further use in prediction.
4. Model Selection: After clusters are created, we find the best model for each cluster. We are using two algorithms, “Random Forest” and “XGBoost”. For each cluster, both the algorithms are passed with the best parameters derived from GridSearch. We calculate the AUC scores for both the models and select the model with the best score. Similarly, the model is selected for each cluster. All the models for every cluster are saved for use in prediction.

Prediction Data Validation

Client will send the data in multiple set of files in batches at a specified location. Data will contain wafer name & 590 columns of different sensor values for each wafer.

Apart from prediction files, we also require a “schema” file from client which contains all the relevant information about the training files such as:

1. Name Validation: We validate the name of the files based on the given name in the schema file. We have created a regex pattern as per the name specified in schema file. After validating the pattern in the name, we check for the length of data in the file name as well as the length of time in the file name. if all the values are as per the requirement, we move such files to “Good_Data” directory else move such files to “Bad_Data”
2. Number of Columns: We validate the number of columns present in the files, and if it doesn’t match with the value given in the schema file, then the file is moved to “Bad_Data”
3. Name of Columns: The name of the columns is validated and should be the same as given in the schema file. If not, then the file is moved to “Bad_Data”
4. The datatype of columns: The datatype of columns is given in the schema file. This is validated when we insert the files into database. If the datatype is wrong, then move the file to “Bad_Data”.
5. Null Values in Column: If any of the columns in a file have all the values as NULL or missing, we discard such a file and move it to “Bad_Data”

Prediction Data Insertion

1. Database Creation & Connection: create a database named as “prediction_waferdb”. If the database is already created, open the connection to the database.
2. Collection Creation in Database: Collection with name – “prediction_waferCollection” create in database “prediction_waferdb” and insert all the data files from “Good_Data”

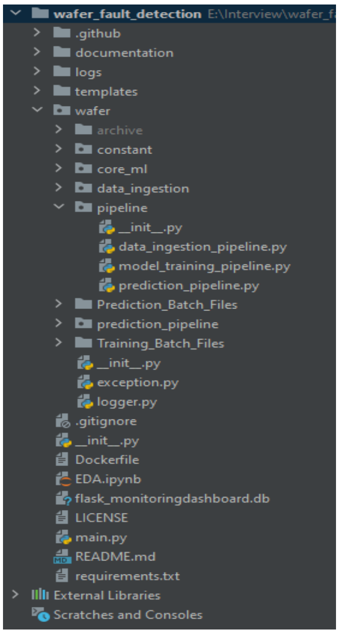
Prediction Result

1. Data Export: Export the data from database “prediction_waferdb” from collection “prediction_waferCollection”in CSV file
2. Data Preprocessing:
    a. Check for NULL Value: If null value(s) present, impute the null values using KNN imputer.
    b. Columns with Zero Standard Deviation: Identify such columns and drop them as these columns do not contribute in model training.
3. Clustering: it’s time to load the KMeans model created during model training and find the cluster for each record.
4. Prediction: Load all the models saved for each cluster and based on the cluster number pass the data for prediction purpose.
5. Once we are ready with the predictions for each record export the predictions along with the wafer number in a CSV file and return the export location

Codebase

Repository over GitHub: https://github.com/bhagwat-chate/wafer_fault_detection.git

The folder structure for the wafer fault detection project:



requirements.txt file consist of the necessary packages list to deploy the app

```
APScheduler==3.9.1.post1
certifi==2022.12.7
charset-normalizer==3.0.1
click==8.1.3
colorama==0.4.6
colorhash==1.2.1
configparser==5.3.0
contourpy==1.0.6
cycler==0.11.0
dnspython==2.2.1
Flask==2.2.2
Flask-Cors==3.0.10
Flask-MonitoringDashboard==3.1.1
fonttools==4.38.0
from-root==1.3.0
greenlet==2.0.1
idna==3.4
importlib-metadata==6.0.0
itsdangerous==2.1.2
Jinja2==3.1.2
joblib==1.2.0
kiwisolver==1.4.4
kneed==0.8.2
MarkupSafe==2.1.2
matplotlib==3.6.3
numpy==1.24.1
packaging==23.0
pandas==1.5.2
Pillow==9.4.0
protobuf==3.20.1
psutil==5.9.4
```

main.py is the entry point for the application, where the flask server starts. Here we will be decoding a base64 to an image, and then we will be making predictions.

main.py prediction_pipeline.py

```
from wafer.pipeline.data_ingestion_pipeline import TrainingDataPipeline
from wafer.pipeline.model_training_pipeline import Model_Training_Pipeline
from wafer.pipeline.prediction_pipeline import Prediction_Pipeline
from wafer.logger import logging
from wafer.exception import WaferException
import sys, os, json, requests
import flask_monitoringdashboard as dashboard

from wsgiref import simple_server
from flask import Flask, request, render_template
from flask import Response
import os
from flask_cors import CORS, cross_origin
import json

os.putenv('LANG', 'en_US.UTF-8')
os.putenv('LC_ALL', 'en_US.UTF-8')

app = Flask(__name__)
dashboard.bind(app)
CORS(app)
```

wafer/pipeline/data_ingestion_pipeline.py – performs all the data ingestion operations discussed above and store the data in database for training purpose.

```python
from wafer.data_ingestion.rawValidation import Raw_Data_Validation
from wafer.data_ingestion.dataTransformation import Data_Transform
from wafer.data_ingestion.dbOperation import DB_Operation
from wafer.logger import logging
from wafer.exception import WaferException
import sys

class TrainingDataPipeline:

    def __init__(self):
        self.raw_data_validation = Raw_Data_Validation("wafer/Training_Batch_Files")
        self.data_transform = Data_Transform()
        self.dbops = DB_Operation()

    def train_data_validation(self):

        """
        Method Name: manualRegexCreation
        Description: This method contains a manually defined regex based on the "FileName"
        This Regex is used to validate the filename of the training data.
        Output: Regex pattern
        On Failure: None
```

wafer/pipeline/model_training_pipeline.py – performs all the preprocessing, clustering, model training, model validation and dump the model operations.

```python
from sklearn.model_selection import train_test_split
from wafer.core_ml.data_preprocessing import Preprocessor
from wafer.core_ml.clustering import KMeansClustering
from wafer.core_ml.model_tuner import Model_Finder
from wafer.core_ml.file_methods import File_Operations
from wafer.logger import logging
from wafer.exception import WaferException
import sys
import pandas as pd


class Model_Training_Pipeline:
    def __init__(self):
        self.col_to_drop = None
        self.data = None
        logging.info("### Model Training Pipeline Initiated ###")
        self.path = "wafer/data_ingestion/Data_Export/Training_data.csv"
        self.preprocessor = Preprocessor()

    def train_model(self, path):
        try:
            self.data = self.preprocessor.get_data(self.path)
```

wafer/pipeline/prediction_pipeline.py – responsible for performing the prediction raw data validation, load to DB, export from DB, load clustering model, respective cluster prediction models, perform predictions and export

```python
from wafer.prediction_pipeline.predict_from_model import Predict_from_Model
from wafer.prediction_pipeline.prediction_DB_operation import Prediction_DB_Operation
from wafer.prediction_pipeline.prediction_data_validation import Prediction_Data_Validation
from wafer.prediction_pipeline.prediction_data_transformation import Prediction_Data_Transformation
from wafer.core_ml.file_methods import File_Operations
from wafer.logger import logging
from wafer.exception import WaferException
import sys
import pandas as pd
from flask import jsonify


class Prediction_Pipeline:

    def __init__(self):
        self.predict_from_model_obj = Predict_from_Model()
        self.prediction_DB_operation_obj = Prediction_DB_Operation()
        self.prediction_data_validation_obj = Prediction_Data_Validation()
        self.prediction_data_transformation_obj = Prediction_Data_Transformation()
        self.file_operations_obj = File_Operations()
```

wafer/logger.py – During the application execution each and every activity is recorded with the help of python logging class.

logger.py

```python
import logging
import os
from datetime import datetime
import os

LOG_FILE = f"{datetime.now().strftime('%m_%d_%Y_%H_%M_%S')}.log"

logs_path = os.path.join(os.getcwd(), "logs", LOG_FILE)

os.makedirs(logs_path, exist_ok=True)

LOG_FILE_PATH = os.path.join(logs_path, LOG_FILE)

logging.basicConfig(
    filename=LOG_FILE_PATH,
    format="[ %(asctime)s ] %(lineno)d %(name)s - %(levelname)s - %(message)s",
    level=logging.INFO,
)
```

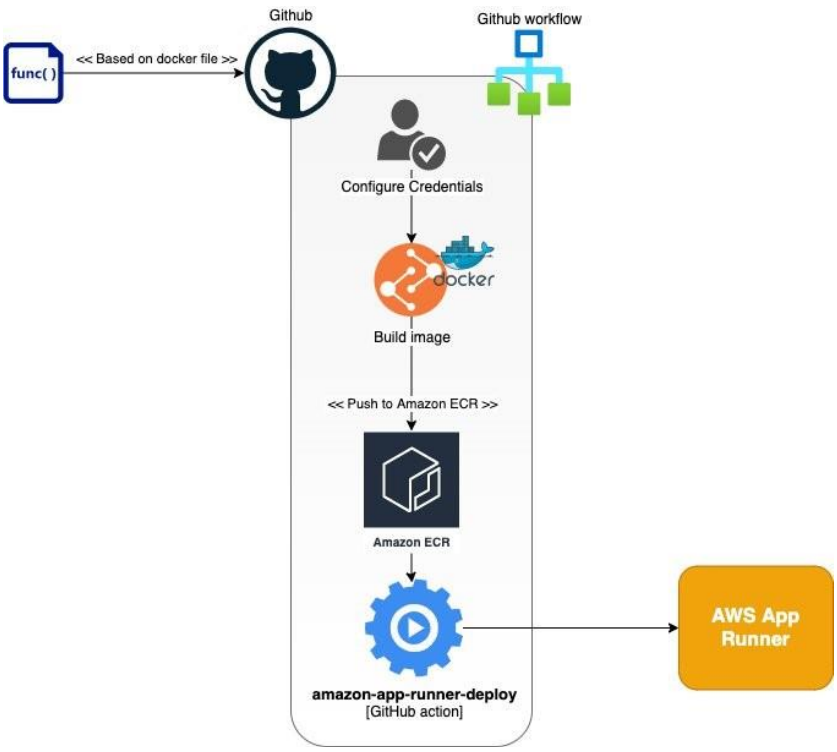
wafer/exception.py – in all the code base the exceptions are handled carefully.

exception.py

```python
import sys


bhagwat-chate
def error_message_detail(error, error_detail: sys):
    _, _, exc_tb = error_detail.exc_info()

    file_name = exc_tb.tb_frame.f_code.co_filename

    error_message = "Error occurred python script name [{0}] line number [{1}] error message [{2}]".format(
        file_name, exc_tb.tb_lineno, str(error)
    )

    return error_message


bhagwat-chate
class WaferException(Exception):
    bhagwat-chate
    def __init__(self, error_message, error_detail:sys):
        """
        :param error_message: error message in string format
        """
        super().__init__(error_message)
```

Application Deployment

The deployment pipeline: Automated CI/CD pipeline with GitAction + AWS App runner



High-level view of the end-to-end application deployment flow

To achieve the above mentioned architecture, we have created the GitHub workflow at:

.github/workflows/main.yaml – Contains all the instructions to run the workflow

main.yaml

```yaml
name: workflow

on:
  push:
    branches:
      - main
    paths-ignore:
      - 'README.md'

permissions:
  id-token: write
  contents: read

jobs:
  integration:
    name: Continuous Integration
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v3
```

Deployment Methodologies:

Here, we are going to deploy the application over AWS, the required steps are:

- To run the app at local:
  a. Clone the repository with command

```
(base) E:\app_test>git clone https://github.com/bhagwat-chate/wafer_fault_detection.git
```

  b. Create new virtual environment with command

```
(base) E:\app_test>conda create -n wafer_app_test python==3.8.15 -y
Collecting package metadata (current_repodata.json): done
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done
```

  c. Activate the environment

```
(base) E:\app_test>activate wafer_app_test

(wafer_app_test) E:\app_test>
```

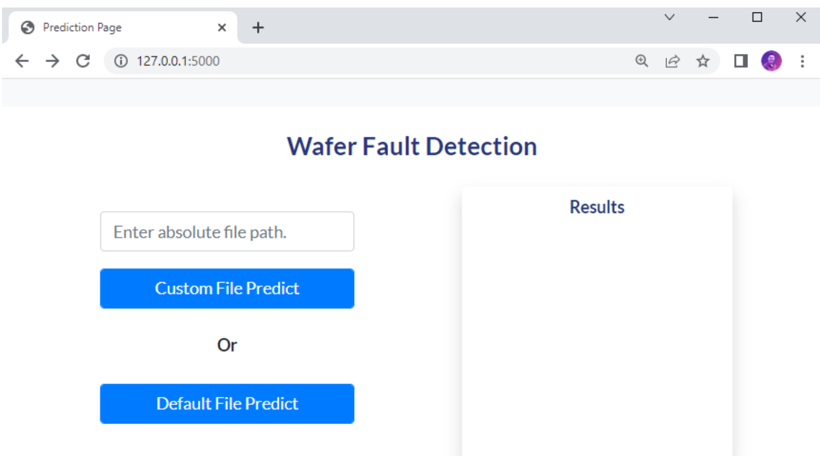
  d. Go to the newly downloaded repository and run command. It will install all the required packages

```
(wafer_app_test) E:\app_test>cd wafer_fault_detection

(wafer_app_test) E:\app_test\wafer_fault_detection>pip install -r requirements.txt
Collecting APScheduler==3.9.1.post1
```

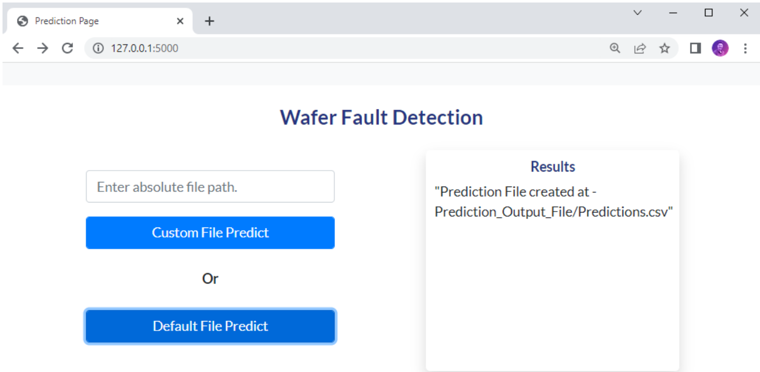
  e. To run the application

```
(wafer_app_test) E:\app_test\wafer_fault_detection>python main.py
Scheduler started
 * Serving Flask app 'main'
 * Debug mode: off
```

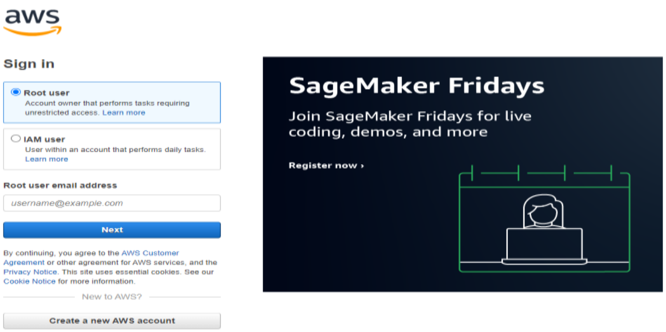
  f. Go to the browser and run: 127.0.0.1:5000
     Prediction home page-



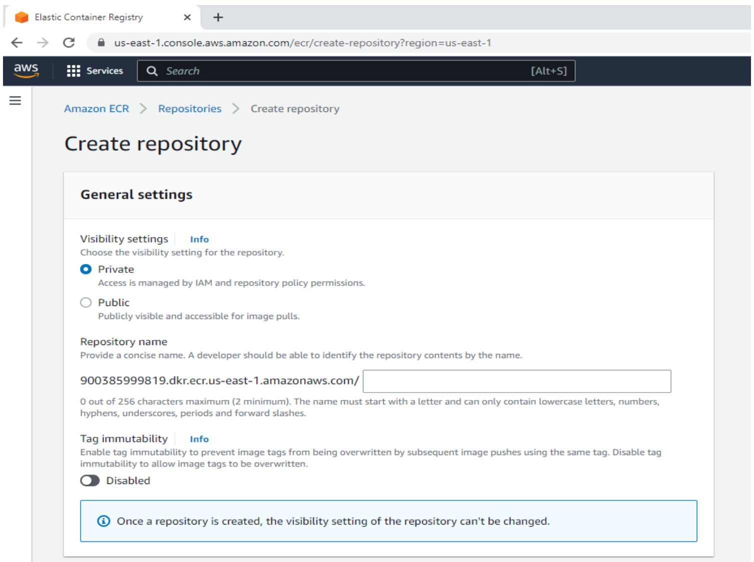
Provide the files directory path or proceed with default path setup for the predictions.



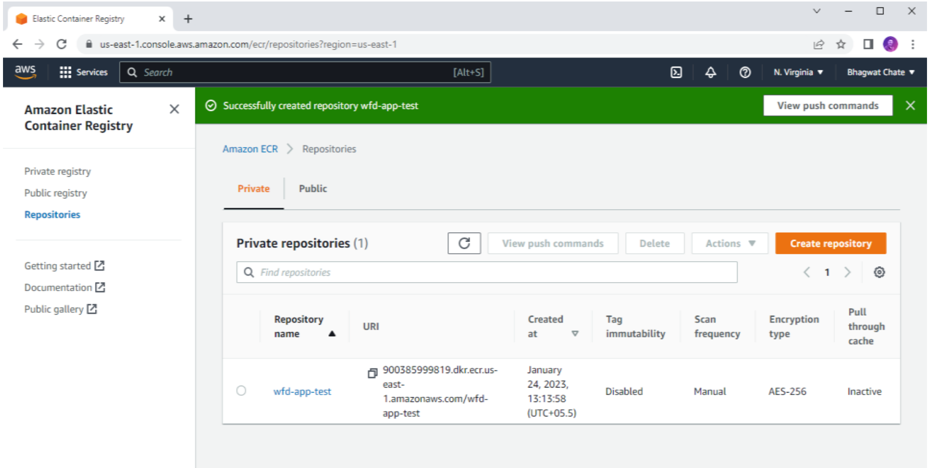
- Login the AWS – go to the AWS console click here, create the account



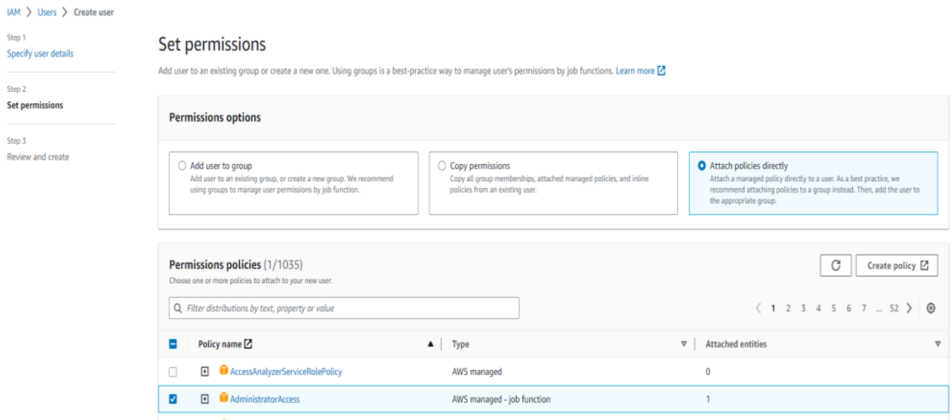
a. Create the AWS repository: enter repo name appropriate



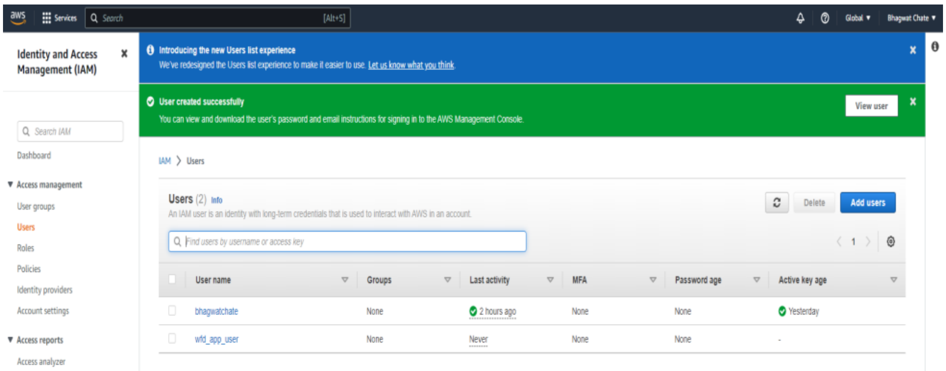
The newly created repository is:



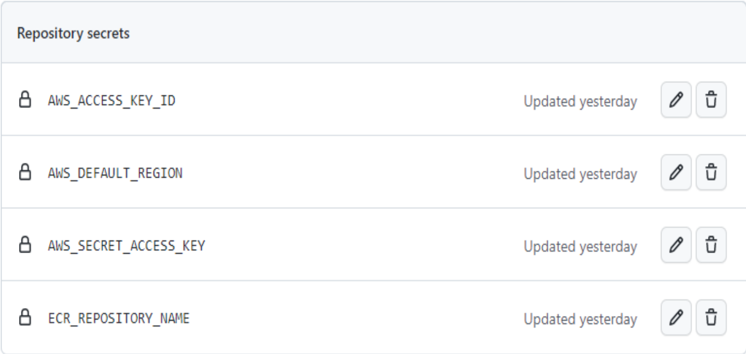
b. Create policy for the application



c. New user – **wfd_app_user** added in IAM



d. Configure the AWS secrets in GitHub



e. Configure AWS to local system (update the secrets such as):
AWS_ACCESS_KEY_ID
AWS_DEFAULT_REGION
AWS_SECRET_ACCESS_KEY
ECR_REPOSITORY_NAME

```
(wafer_app_test) E:\app_test\wafer_fault_detection>aws configure
AWS Access Key ID [****************FA75]: [redacted]
AWS Secret Access Key [****************Kz5b]: [redacted]
Default region name [ep-northeast-1]: [redacted]
Default output format [json]: [redacted]

(wafer_app_test) E:\app_test\wafer_fault_detection>
```

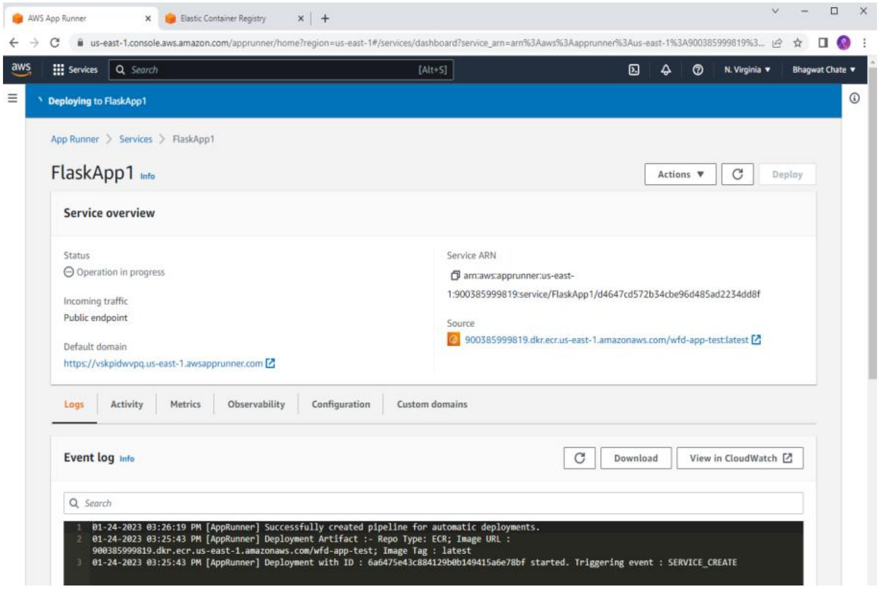
f. Build docker image at local

```
(wafer_app_test) E:\app_test\wafer_fault_detection>docker build -t wfd_app .
```

g. Go to the AWS App Runner service and execute **Push commands for wfd-app-test** one by one (These commands push the docker image from local to AWS ECR)

```
(wafer_app_test) E:\app_test\wafer_fault_detection>aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 9003
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
 => [3/5] WORKDIR /app
 => [4/5] RUN pip3 install --upgrade pip
 => [5/5] RUN pip3 install -r requirements.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:5b017523e63be214664d43d74aa824dd50fb14c55b3df3a5642f27e583562029
 => => naming to docker.io/library/wfd-app-test

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

(wafer_app_test) E:\app_test\wafer_fault_detection>docker tag wfd-app-test:latest 900385999819.dkr.ecr.us-east-1.amazonaws.com/wfd-app-test:latest

(wafer_app_test) E:\app_test\wafer_fault_detection>docker push 900385999819.dkr.ecr.us-east-1.amazonaws.com/wfd-app-test:latest
The push refers to repository [900385999819.dkr.ecr.us-east-1.amazonaws.com/wfd-app-test]
404b6548401: Pushing [===>                                              ]  67.19MB/954.1MB
4ab94cd792a4: Pushed
```

h. Configure the AWS App Runner and run



Future Enhancements:

- Develop data ingestion pipeline for streaming data
- Implement scheduler for streaming data inputs for prediction
- Integrate prediction result in analytics tool such as Power BI / Tableau / QlikSense
- Centralize constants variable repository in codebase
- Implement AWS S3 service for data storage

Developer:

Mr. Bhagwat Chate

References:

Website:

- https://en.wikipedia.org/wiki/Wafer_(electronics)
- https://developers.google.com/machine-learning/guides/rules-of-ml
- https://stackoverflow.com/
- https://scikit-learn.org/stable/
- https://www.geeksforgeeks.org/
- https://machinelearningmastery.com/
- https://docs.aws.amazon.com/apprunner/latest/dg/what-is-apprunner.html
- https://github.com/sauravraghuvanshi/Udacity-programming-for-Data-Science-With-Python-Nanodegree/blob/master/Project-3/Git%20Commands%20Documentation.pdf

- https://yamari.co.in/industries/semiconductor/wafer-sensor
- https://www.thermo-electric.com/products/silicon-wafers/

Book:

- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition
- Mathematics for Machine Learning by A. Aldo Faisal, Cheng Soon Ong, and Marc Peter Deisenroth