W11: TA Session

Quality, Requirements, Architecture, and Microservices.

Objective:

Implementing a microservice using the Python Flask framework on an Ubuntu virtual machine to serve a machine learning prediction model.

Context:

For this case study we are going to work with the following dataset:
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. n the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

Attribute Information:

- ID number
- Diagnosis (M = malignant, B = benign)

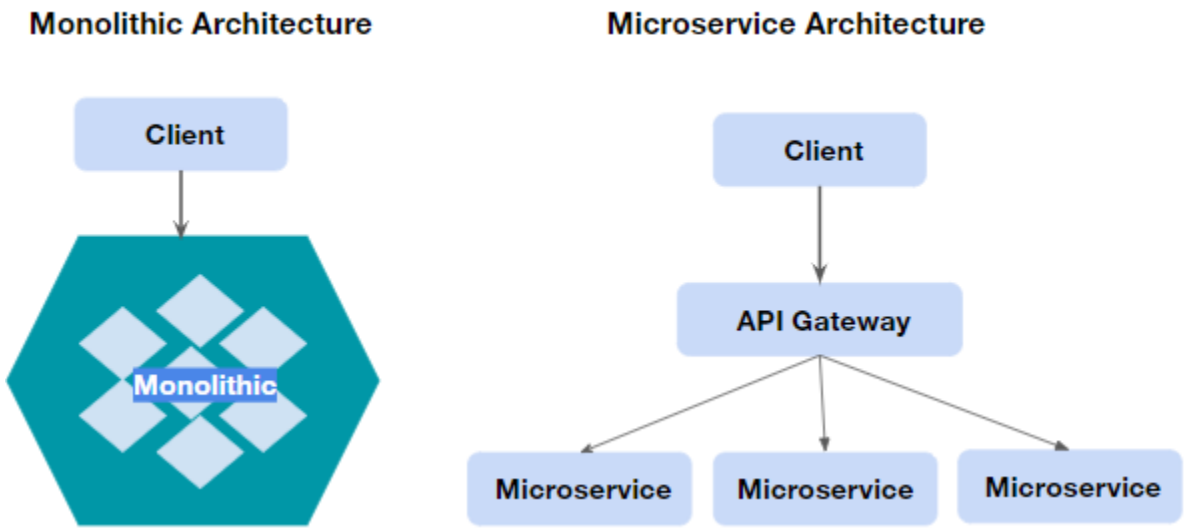
Ten real-valued features are computed for each cell nucleus:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area.
- smoothness (local variation in radius lengths)
- compactness (perimeter^2 / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

Machine Learning Task:

To build a machine learning model to predict whether the cancer type is Malignant or Benign.

Microservice Implementation:
Our main goal is to design a microservice. Which are standalone, manageable, tested, loosely coupled services that frequently only have one actual use case or function. Microservices are a method for creating a single application as a collection of smaller services, each of which runs in its own process and communicates with simple tools, frequently an HTTP resource API.



Python Flask ML Application as a microservice on an Ubuntu virtual machine.

1. First we must host a Ubuntu virtual machine using the Oracle VM Virtual box.
2. Then we must create the endpoints for subsequently interacting with the client via the HTTP protocol, train and save a machine learning model, and wrap it in a Flask web application.
3. Finally we will run and test our application using some example calls.

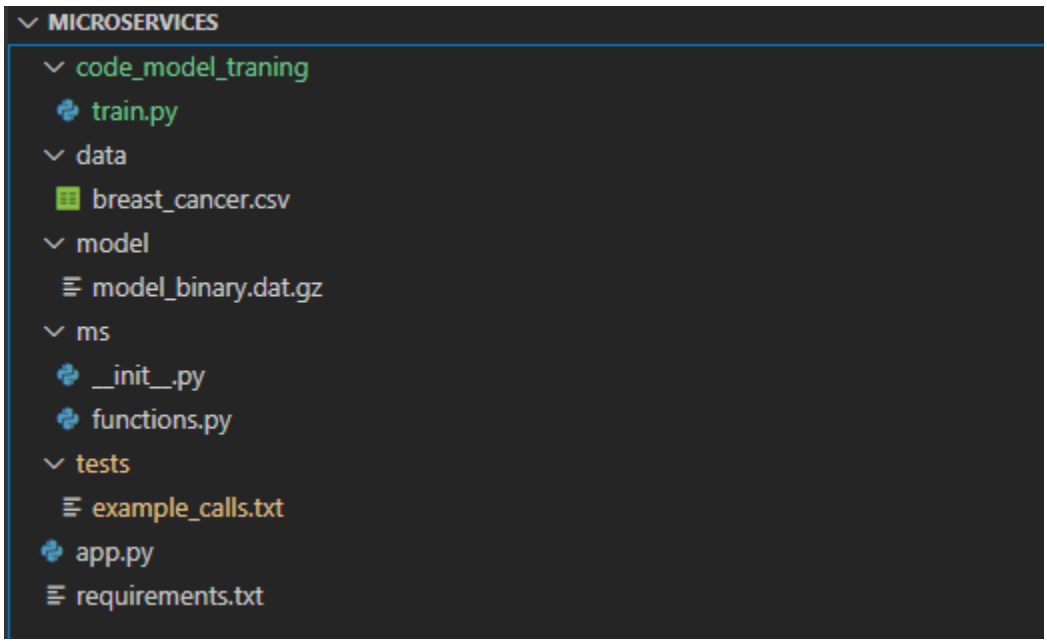
Steps and Instructions:

- Host a Ubuntu Virtual Machine using Oracle VM Virtual Box.
- Set up Visual Studio code on Ubuntu VM.
  For Setup refer to this video https://www.youtube.com/watch?v=N_Ve4iAzxq8
- Download the python version 3.10, you may refer to the below commands,
  sudo apt update
  sudo apt install software-properties-common
  sudo add-apt-repository ppa:deadsnakes/ppa
  sudo apt install python3.10
  Verification of the installation was successful. Python3 –version

Clone the GitHub repository:

- From Visual Studio code clone the below github repository.
  https://github.com/Vikas098766/Microservices.git

Let's look the project structure below;



Virtual Environment:

First we need to create a virtual environment for the cloned github project, to keep track of every dependency, it is also useful to use an explicit version of Python.

- To set up a Python 3 virtual environment, navigate to your project folder on your terminal and type the following command → python3 -m venv venv
  This will create a new virtual environment named venv using the version of Python 3 that you have installed on your system.
- Next, you need to activate the virtual environment by sourcing the activation script: command → source venv/bin/activate
- After executing this command, your prompt will change to indicate that you're now operating from within the virtual environment.

Now with the virtual environment we can install the dependencies written in requirements.txt:

Command → pip install -r requirements.txt

Model Training and saving the model.

After we have installed all the dependencies we can now run the script in **code_model_training/train.py,** this script takes the input data and outputs a trained model and a pipeline for our web service.

Command → python code_model_training/train.py

Web Application.

Finally we can test our web application by running:

Command → flask run -p 5000

Testing the application and making predictions:

The below command will run the development server locally and listen to port 5000 where we can test our application! The folder ***/tests*** contain some example calls to test that our application is up and running:

Example call:

```
curl -X GET http://localhost:5000/info
```

The service should respond:

{“name”:”Breast Cancer Wisconsin (Diagnostic)”,”version”:”v1.0.0"}

Example call:

POST method predict:

```
curl -d '[{"radius_mean": 17.99, "texture_mean": 10.38, "perimeter_mean": 122.8, "area_mean": 1001.0, "smoothness_mean": 0.1184, "compactness_mean": 0.2776, "concavity_mean": 0.3001, "concave points_mean": 0.1471, "symmetry_mean": 0.2419, "fractal_dimension_mean": 0.07871, "radius_se": 1.095, "texture_se": 0.9053, "perimeter_se": 8.589, "area_se": 153.4, "smoothness_se": 0.006399, "compactness_se": 0.04904, "concavity_se": 0.05373, "concave points_se": 0.01587, "symmetry_se": 0.03003, "fractal_dimension_se": 0.006193, "radius_worst": 25.38, "texture_worst": 17.33, "perimeter_worst": 184.6, "area_worst": 2019.0, "smoothness_worst": 0.1622, "compactness_worst": 0.6656, "concavity_worst": 0.7119, "concave points_worst": 0.2654, "symmetry_worst": 0.4601, "fractal_dimension_worst": 0.1189}]' \
    -H "Content-Type: application/json" \
    -X POST http://0.0.0.0:5000/predict
```

The service should respond:

{“label”:”M”,”prediction”:1,”status”:200}