Lucy Nowacki

quantlucy@gmail.com

LISUM32

05/05/2024

Submitted to Github: https://github.com/LucyNowacki/AzureCancerDetector

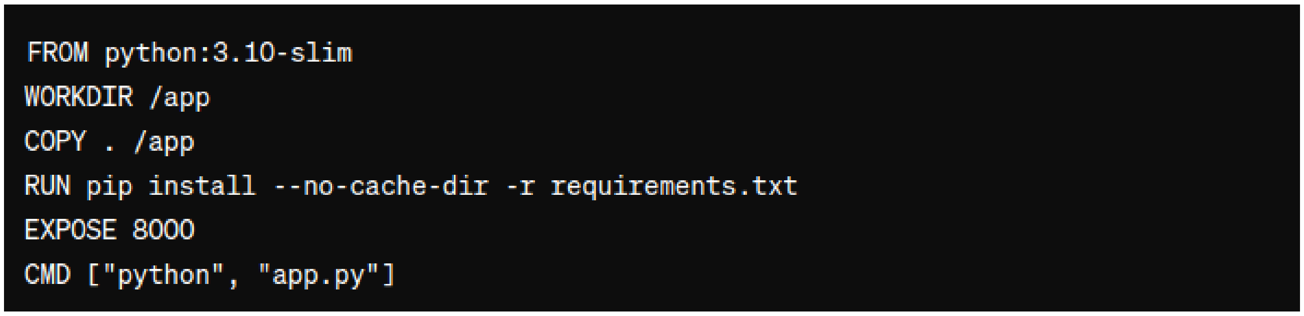
App is working at: https://breastcancerdetector.azurewebsites.net/

Detailed Guide: Flask Application Deployment on Azure

1. Setting Up Flask Application

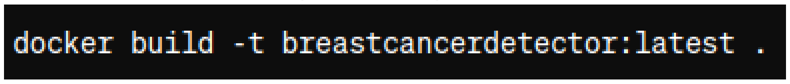
- **Write the Application**: Create a Flask application (**app.py**). This script initializes our Flask app and defines routes to handle requests.
- **Manage Dependencies**: List all necessary Python packages including Flask in a file named **requirements.txt**. This file is used by Docker to install the dependencies in the container.

2. Containerizing the Application with Docker

- **Create a Dockerfile**: This file 'MyDockerfile.dockerfile' contains the instructions for building the Docker image.
  - **Base Image**: Start with a Python image, **python:3.10-slim**, which is a lightweight version containing Python 3.10.
  - **Working Directory**: Set **/app** as the working directory where the application files will reside within the container.
  - **Copying Files**: Copy your application files into the container.
  - **Install Dependencies**: Use **pip install** to install the packages listed in **requirements.txt**.
  - **Port Exposure**: Expose port 8000, which is the port that Flask will run on by default.
  - **Run Command**: Define the command to run the Flask application on container startup.
  - 

```
FROM python:3.10-slim
WORKDIR /app
COPY . /app
RUN pip install --no-cache-dir -r requirements.txt
EXPOSE 8000
CMD ["python", "app.py"]
```

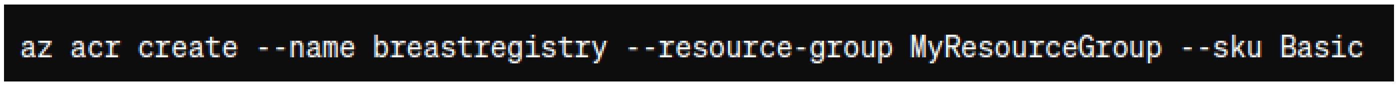
- **Build the Docker Image**: Run the command
  docker build -t your-image-name:tag .



```
docker build -t breastcancerdetector:latest .
```

3. Pushing the Docker Image to Azure Container Registry

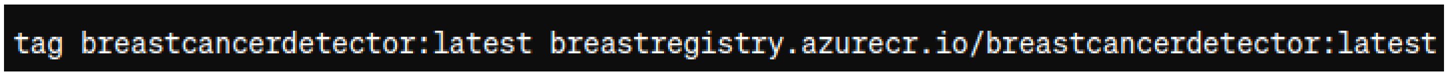
- **Create Azure Container Registry** (if not already done): You can create this through the Azure portal or use the Azure CLI:
- az acr create --name MyRegistry --resource-group MyResourceGroup --sku Basic



```
az acr create --name breastregistry --resource-group MyResourceGroup --sku Basic
```

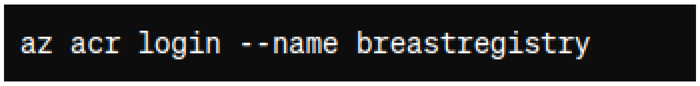
--name breastregistry: The name of your Azure Container Registry.
--resource-group MyResourceGroup: The resource group under which the registry will be created.

- **Tag the Docker Image for ACR**: Tag your image to correspond with your ACR's login server address:
  docker tag your-image-name:tag myregistry.azurecr.io/your-image-name:tag



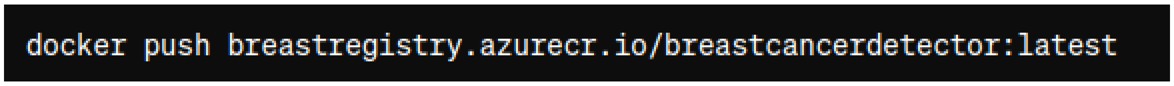
```
tag breastcancerdetector:latest breastregistry.azurecr.io/breastcancerdetector:latest
```

- **Authenticate with ACR**: Log in to your Azure Container Registry using Docker. You might need to enable admin user on ACR or use service principals:
  az acr login --name myregistry



```
az acr login --name breastregistry
```

- **Push the Image to ACR**: Push your tagged image to the registry:
- docker push myregistry.azurecr.io/your-image-name:tag



```
docker push breastregistry.azurecr.io/breastcancerdetector:latest
```

4. Deploying on Azure App Services

After following the steps you've described on the image for containerizing your application with Docker and pushing it to an Azure Container Registry, you're well-prepared to deploy your application to Azure App Services. Here's a detailed step-by-step approach you can follow to deploy your Docker container to Azure App Services efficiently and effectively:

- **Create or Update an App Service Plan**

  You need an App Service Plan which defines the region, size, and features of the web server farm that hosts your app. You can either use an existing plan or create a new one.

```
az appservice plan create --name MyServicePlan --resource-group MyResourceGroup --sku B1 --is-linux --
location "UK South"
```

Create a Web App within the App Service Plan
Create a new Web App to host your Docker container. You can do this using the Azure CLI to specify the use of the Docker image you pushed to your Azure Container Registry

```
az webapp create --resource-group MyResourceGroup --plan MyServicePlan --name BreastCancerDetector --deployment-container-image-name
```

This command sets up a web app named "BreastCancerDetector" using the Docker image from your ACR.

- **Create a Web App within the App Service Plan**
  Create a new Web App to host your Docker container. You can do this using the Azure CLI to specify the use of the Docker image you pushed to your Azure Container Registry.

```
az webapp create --resource-group MyResourceGroup --plan MyServicePlan --name BreastCancerDetector --deployment-container-image-name breastregistry.azurecr.io/breastcancerdetector:latest
```

This command sets up a web app named "BreastCancerDetector" using the Docker image from your ACR.

- **Configure the Web App to use the Docker Image**

  Ensure that your Web App is configured to pull the correct image from your ACR by setting the right configuration settings.

```
az webapp config container set --name BreastCancerDetector --resource-group MyResourceGroup --docker-custom-image-name "breastregistry.azurecr.io/breastcancerdetector:latest" --docker-registry-server-url "https://breastregistry.azurecr.io" --docker-registry-server-user " breastregistry " --docker-registry-server-password "<password>"
```

This command configures the Web App to use the Docker image from your Azure Container Registry. Replace **<username>** and **<password>** with your registry's credentials

- **Set Environment Variables**
  If your application requires environment variables to run (e.g., database URLs, API keys), you can set them using the Azure Portal or Azure CLI.

```
az webapp config appsettings set --name BreastCancerDetector --resource-group MyResourceGroup --settings VAR1='value1' VAR2='value2'
```

Say we have the following variables

- **DOCKER_REGISTRY_SERVER_URL**
- **DOCKER_REGISTRY_SERVER_USERNAME**
- **DOCKER_REGISTRY_SERVER_PASSWORD**

If we were to set similar variables using the **az webapp config appsettings set** command, it might look something like this:

```
az webapp config appsettings set --name BreastCancerDetector --resource-group MyResourceGroup --settings
DOCKER_REGISTRY_SERVER_URL='https://breastregistry.azurecr.io'
DOCKER_REGISTRY_SERVER_USERNAME='breastregistry'
DOCKER_REGISTRY_SERVER_PASSWORD='YourPasswordHere'
```

There can be also another variables like , e.g. API_KEY, etc.

- if you want to list the current application settings (environment variables) for your Azure Web App **BreastCancerDetector** in the resource group **MyResourceGroup**, the specific Azure CLI command would be:

```
az webapp config appsettings list --name BreastCancerDetector --resource-group MyResourceGroup
```

This command will display the current set of configuration settings (environment variables) that are set for your web app in Azure App Services. It helps verify that all required settings are correctly configured and allows you to check their values if needed.

5. **Monitoring Deployment via Azure Portal:**

Deployment Center:

- **Navigate to the Azure Portal.**
- Go to **"App Services"** and select your application named **"BreastCancerDetector"**.
- Click on **"Deployment Center"** from the left navigation pane. This area provides you with the status of the latest deployments, including logs and detailed information about the deployment process.

For general deployment status checks and to monitor specific deployment slots:

- **List all deployment slots**:

```
az webapp deployment slot list --name BreastCancerDetector --resource-group MyResourceGroup
```

- **View the status of a specific deployment slot** (if you have multiple slots, replace **[YourSlotName]** with the actual name of the slot you wish to check):

```
az webapp deployment slot show --name BreastCancerDetector --resource-group MyResourceGroup --
slot [YourSlotName]
```

- **Log Streaming for Real-Time Monitoring:**

For immediate feedback during deployment and for troubleshooting live services:

Set up Log Streaming:

- This feature is enabled via the Azure Portal under "App Service logs" or can be enabled using the Azure CLI.
- To stream logs directly to your CLI, providing instant access to the output from your application, use:

```
az webapp log tail --name BreastCancerDetector --resource-group MyResourceGroup
```

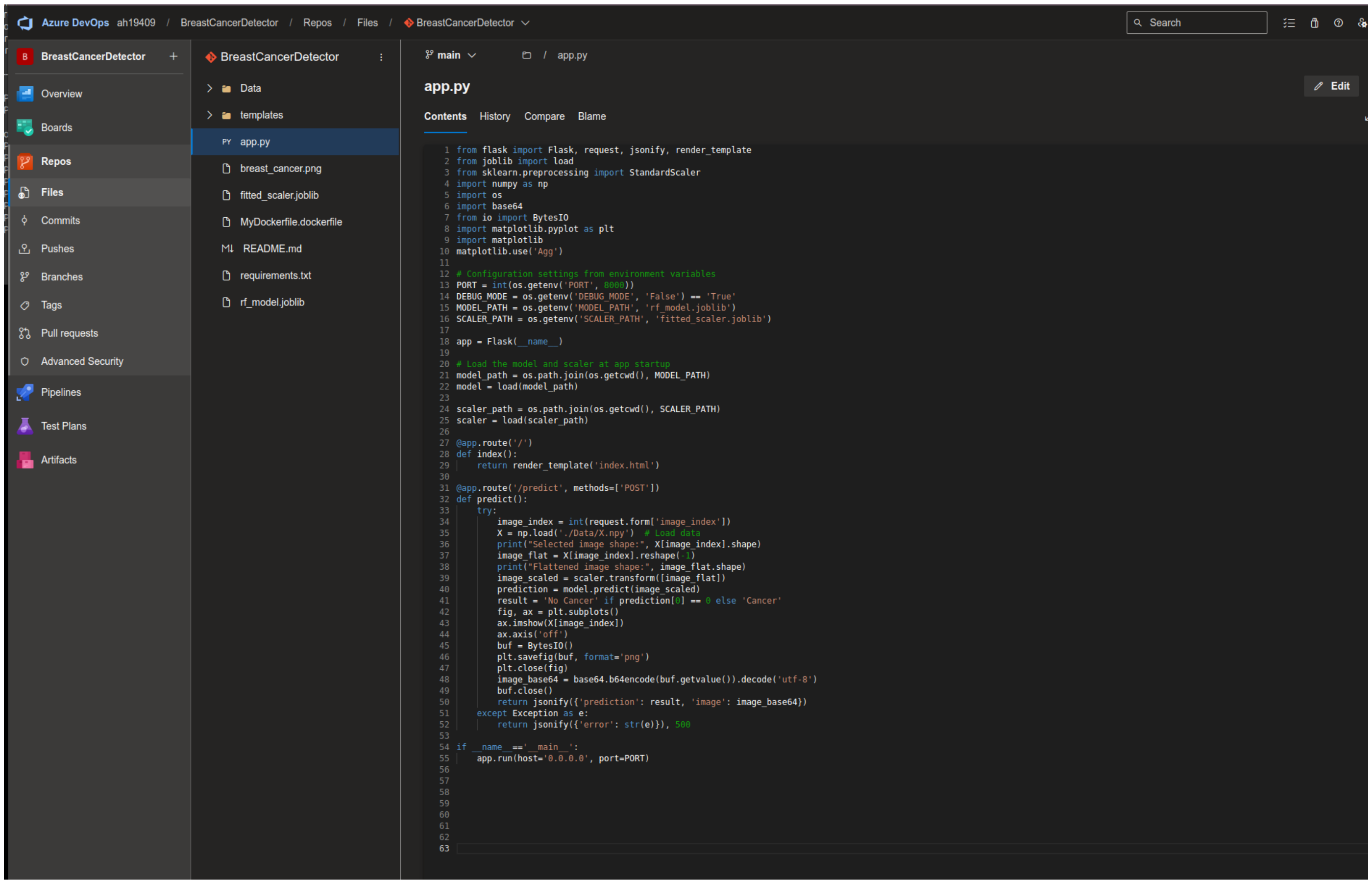
Steps to Enable Log Streaming via Azure Portal:

1. **Navigate to the Azure Portal**.
2. Go to **"App Services"** and select **"BreastCancerDetector"**.
3. Under the **"Monitoring"** section on the left panel, click on **"App Service logs"**.
4. Enable **"Application Logging (Filesystem)"** and set the level of logging (e.g., Information, Warning, Error).
5. Save the configuration to start streaming logs.

By following these steps, you can effectively monitor the deployment and operational status of your Azure App Service, enabling quick diagnostics and insights into your application's performance and health.

Let's look at the screenshots about how the project 'BreastCancerDetector' is nested on Azure.

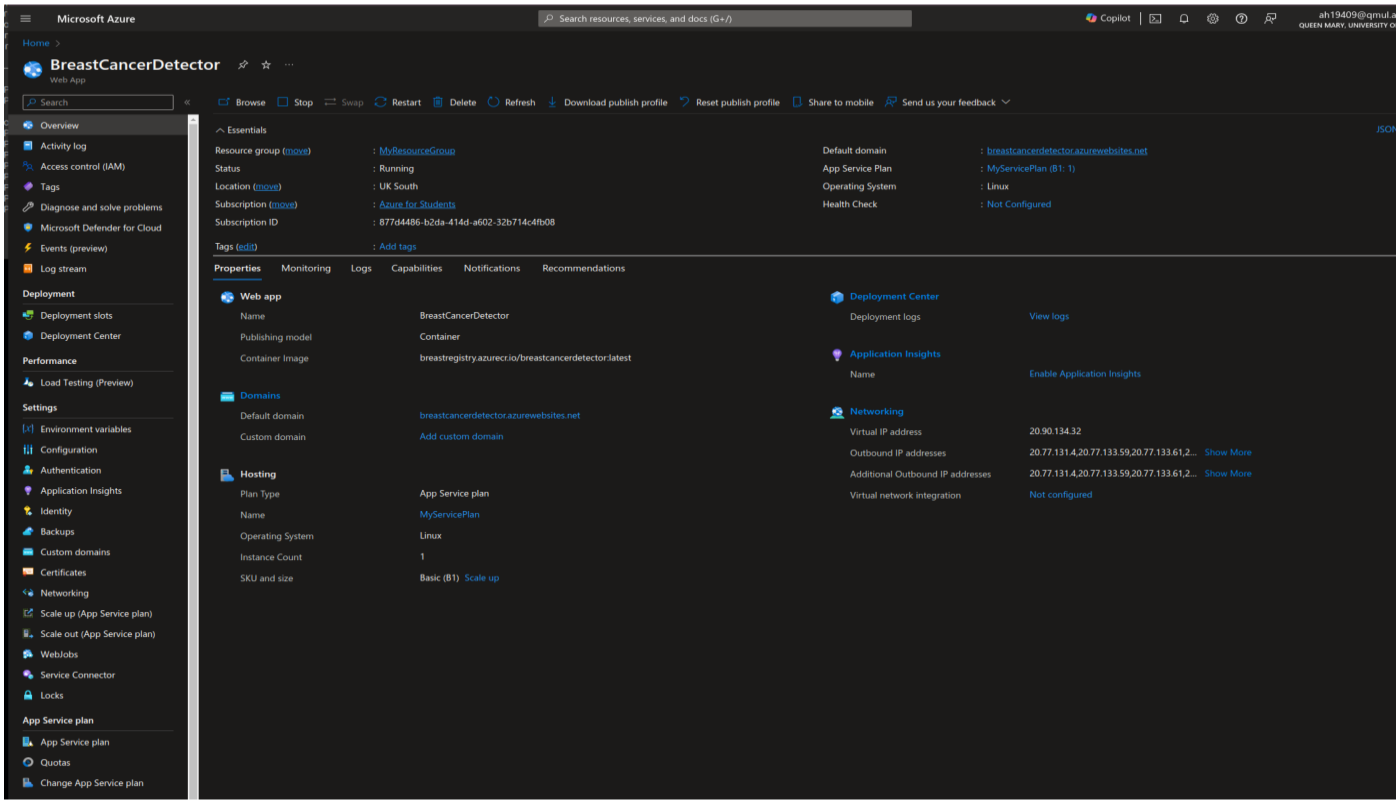
1. After pushing to Azure DevOps



app.py

```
from flask import Flask, request, jsonify, render_template
from joblib import load
from sklearn.preprocessing import StandardScaler
import numpy as np
import os
import base64
from io import BytesIO
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('Agg')

# Configuration settings from environment variables
PORT = int(os.getenv('PORT', 8000))
DEBUG_MODE = os.getenv('DEBUG_MODE', 'False') == 'True'
MODEL_PATH = os.getenv('MODEL_PATH', 'rf_model.joblib')
SCALER_PATH = os.getenv('SCALER_PATH', 'fitted_scaler.joblib')

app = Flask(__name__)

# Load the model and scaler at app startup
model_path = os.path.join(os.getcwd(), MODEL_PATH)
model = load(model_path)

scaler_path = os.path.join(os.getcwd(), SCALER_PATH)
scaler = load(scaler_path)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    try:
        image_index = int(request.form['image_index'])
        X = np.load('./Data/X.npy')  # Load data
        print("Selected image shape:", X[image_index].shape)
        image_flat = X[image_index].reshape(-1)
        print("Flattened image shape:", image_flat.shape)
        image_scaled = scaler.transform([image_flat])
        prediction = model.predict(image_scaled)
        result = 'No Cancer' if prediction[0] == 0 else 'Cancer'
        fig, ax = plt.subplots()
        ax.imshow(X[image_index])
        ax.axis('off')
        buf = BytesIO()
        plt.savefig(buf, format='png')
        plt.close(fig)
        image_base64 = base64.b64encode(buf.getvalue()).decode('utf-8')
        buf.close()
        return jsonify({'prediction': result, 'image': image_base64})
    except Exception as e:
        return jsonify({'error': str(e)}), 500

if __name__=='__main__':
    app.run(host='0.0.0.0', port=PORT)
```

2. After deployment



BreastCancerDetector
Web App

Essentials

Resource group (move)	: MyResourceGroup
Status	: Running
Location (move)	: UK South
Subscription (move)	: Azure for Students
Subscription ID	: 877d4486-b2da-414d-a602-32b714c4fb08
Tags (edit)	: Add tags
Default domain	: breastcancerdetector.azurewebsites.net
App Service Plan	: MyServicePlan (B1: 1)
Operating System	: Linux
Health Check	: Not Configured

Web app

Name	BreastCancerDetector
Publishing model	Container
Container Image	breastregistry.azurecr.io/breastcancerdetector:latest

Domains

Default domain	breastcancerdetector.azurewebsites.net
Custom domain	Add custom domain

Hosting

Plan Type	App Service plan
Name	MyServicePlan
Operating System	Linux
Instance Count	1
SKU and size	Basic (B1) Scale up

Deployment Center

Deployment logs	View logs

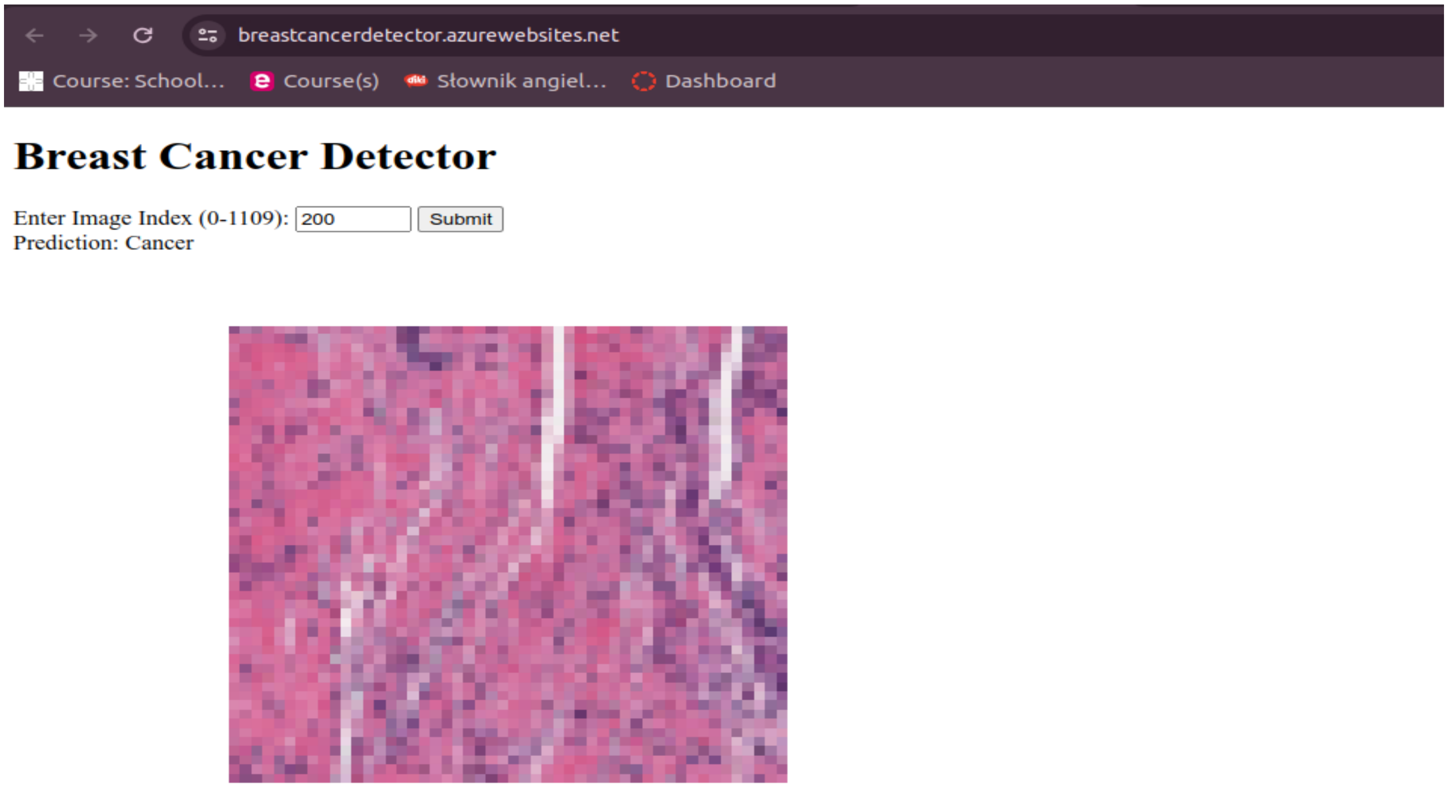
Application Insights

Name	Enable Application Insights

Networking

Virtual IP address	20.90.134.32
Outbound IP addresses	20.77.131.4,20.77.133.59,20.77.133.61,2... Show More
Additional Outbound IP addresses	20.77.131.4,20.77.133.59,20.77.133.61,2... Show More
Virtual network integration	Not configured

3. The working application is available at https://breastcancerdetector.azurewebsites.net/



Breast Cancer Detector

Enter Image Index (0-1109): 200

Prediction: Cancer