Deployed Multi-Disease Prediction WebApp using

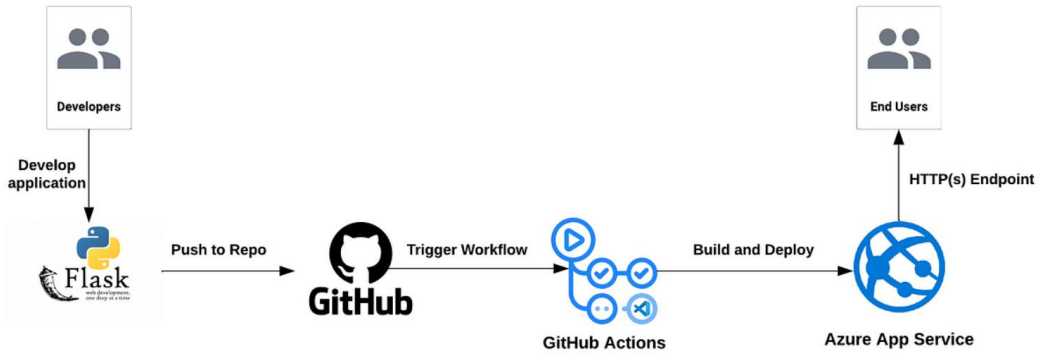
[Azure Web App Service, Virtual Machine, and github Actions]

Problem statement –

Despite the availability of advanced medical technology and resources, the lack of an efficient and organized healthcare system in many parts of the world poses significant challenges for individuals seeking accurate and timely diagnosis of diseases. This is particularly true in rural areas where farmers and other individuals face limited access to specialized healthcare services.

In India, for example, where agriculture is a major occupation, farmers and rural communities often struggle with inadequate healthcare infrastructure and the absence of organized diagnostic facilities. This results in delayed detection and diagnosis of critical diseases such as pneumonia, malaria, diabetes, heart disease, and breast cancer, leading to increased morbidity rates and limited treatment options.

Solution :



To address these challenges, I have developed a WebApp using Azure services, GitHub Actions, Azure Web App service, and Virtual Machine service. This multidisease detection WebApp aims to provide accessible and reliable disease prediction for farmers and individuals in rural areas. By leveraging machine learning algorithms and medical data analysis, the WebApp can assist in the early detection and diagnosis of diseases, empowering individuals to seek appropriate medical care and improve their overall well-being.

Project Link : https://multidisease.azurewebsites.net/

Pre-Requisites:

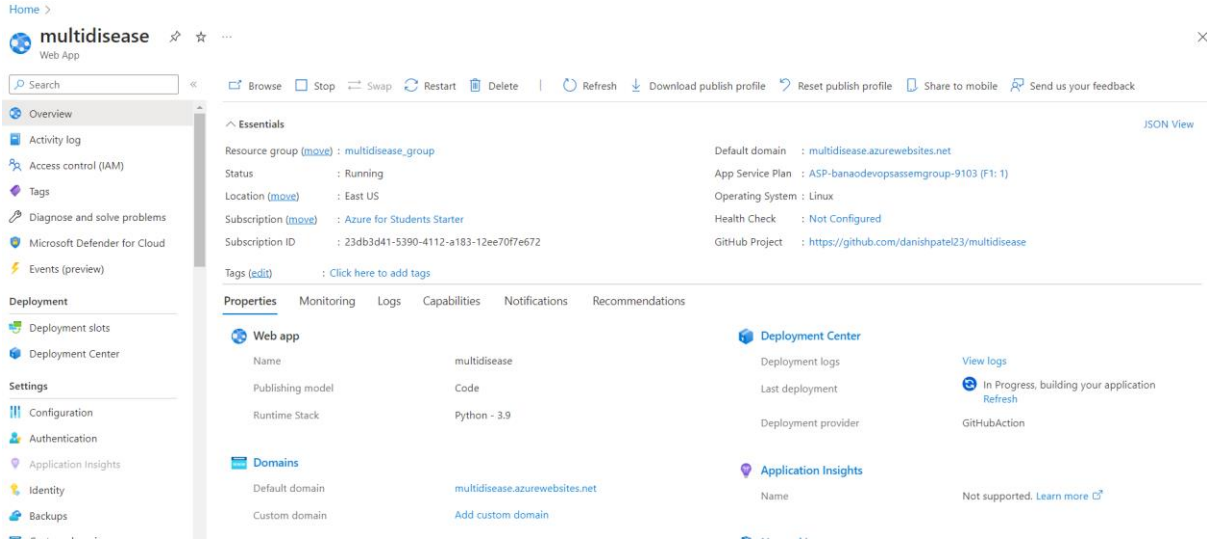
- **You will need an Azure subscription. If you do not have one, you**
- **can get one for free. Click here to create a free Azure account.**
- **Azure Web App services and VM credits .**
- **Make sure you have Python installed and the VS Code Python**

Steps 1:

Create webapp service:

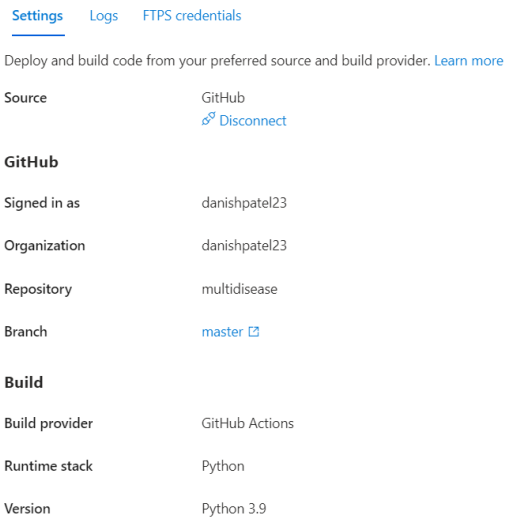
Sign in to the Azure portal, navigate to the App Services section, and click on "Create a resource." Select "Web App" and provide the required details such as the subscription, resource group, and unique app name.

Configure the web app settings, including the runtime stack (Python,), operating system, and deployment options (such as Docker or GitHub integration). Finally, click on "Create" to provision the Azure Web App.

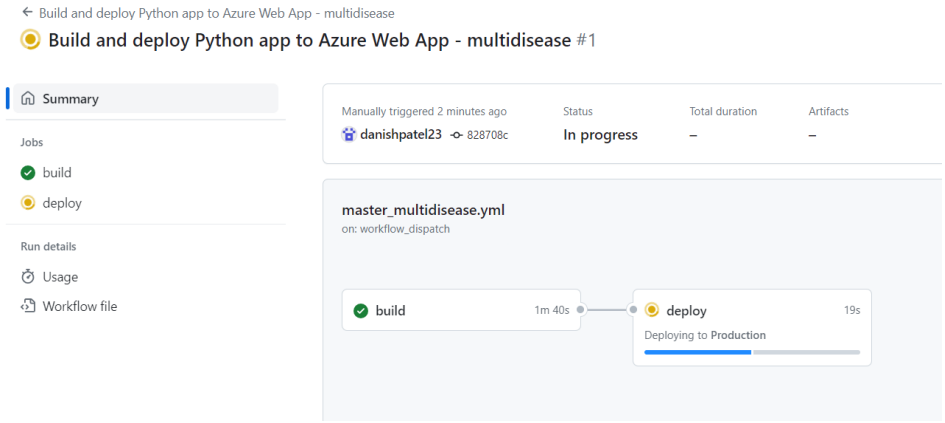


Steps to configure Deployment Center for GitHub Actions in Azure:

1. **Navigate to the Azure portal, open your Azure Web App, and go to the "Deployment Center" section. Select "GitHub Actions" as the source control option.**
2. **Follow the prompts to connect your GitHub account, select the repository containing your web app code, and configure the deployment settings. Specify the branch to deploy from, choose the build workflow file (usually located in the .github/workflows directory), and customize any additional deployment options as needed.**



After saving you will see the workflow file trigger in github actions

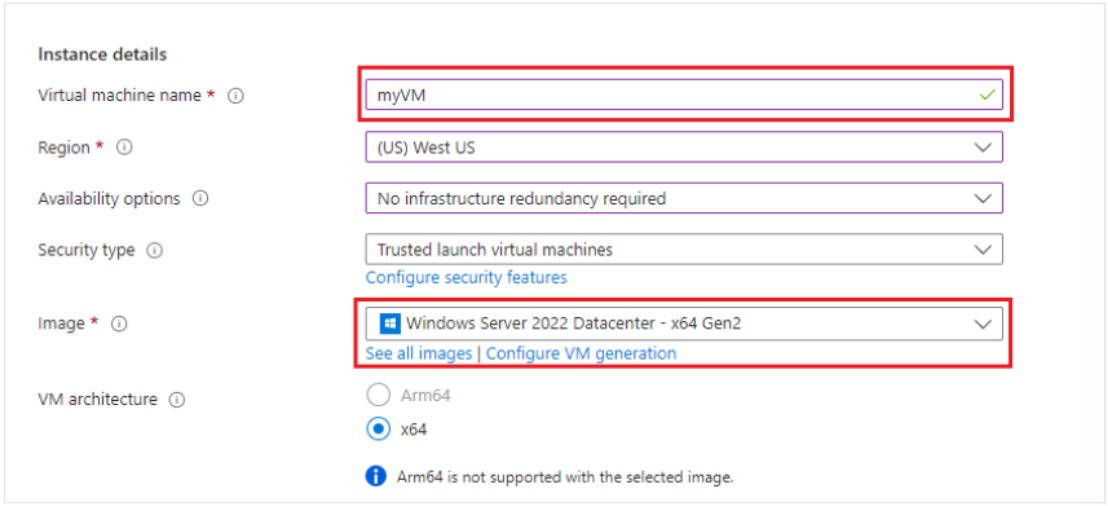


STEP 2 : windows VM creating

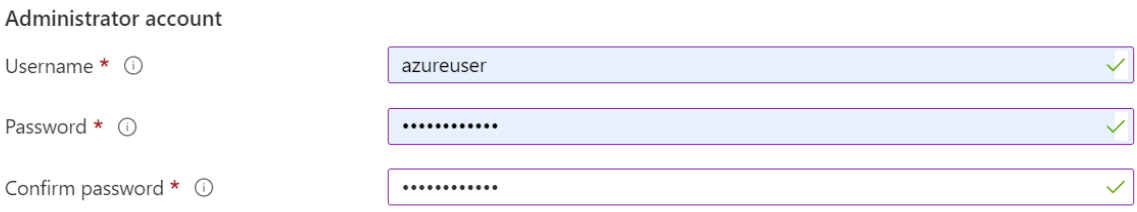
Sign in to the Azure portal, click on "Create a resource," and search for "Windows Server" in the marketplace. Select the desired Windows Server version and click on "Create."

Provide the necessary details for the VM, such as the subscription, resource group, VM name, region, and availability options. Choose the appropriate VM size, specify the username and password for authentication, and configure additional settings like networking, storage, and monitoring. Finally, review and validate the configuration, then click on "Create" to provision the Windows VM in Azure.

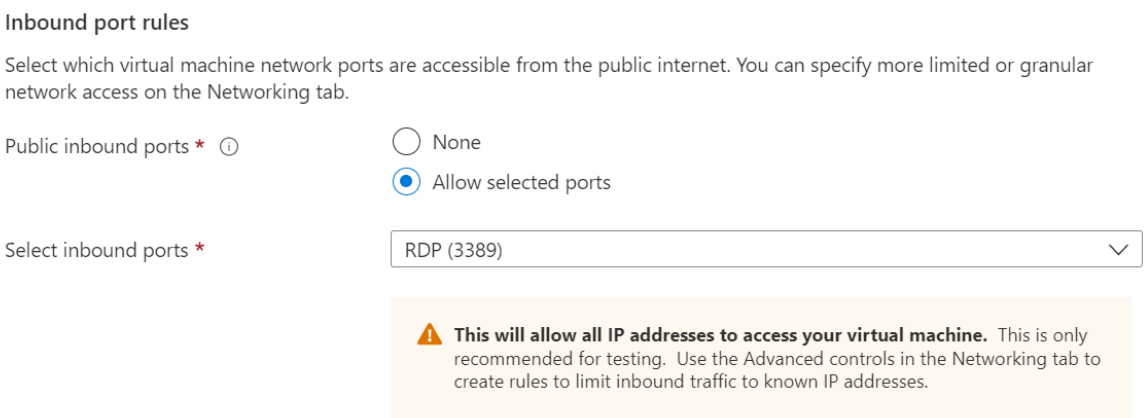
. Under **Instance details**, enter *myVM* for the **Virtual machine name** and choose *Windows Server 2022 Datacenter - Gen 2* for the **Image**. Leave the other defaults.

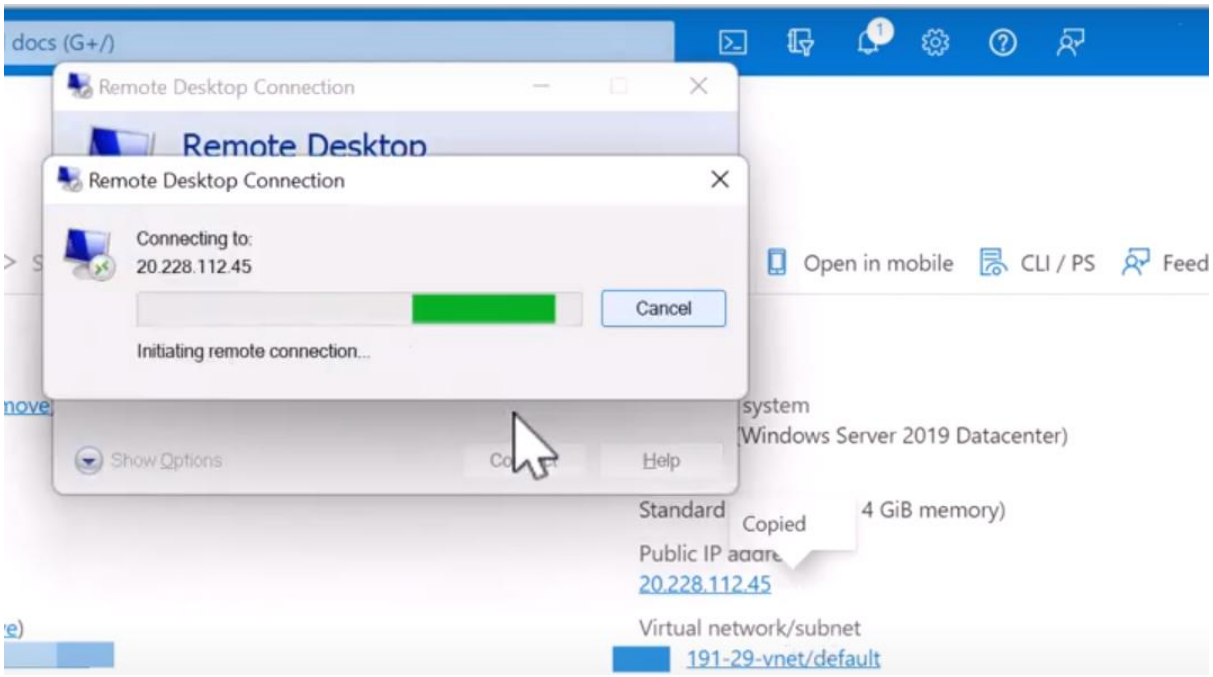


1. Under **Administrator account**, provide a username, such as *azureuser* and a password. The password must be at least 12 characters long and meet the defined complexity requirements.



2. Under **Inbound port rules**, choose **Allow selected ports** and then select **RDP (3389)** and **HTTP (80)** from the drop-down.





Connecting user RDP

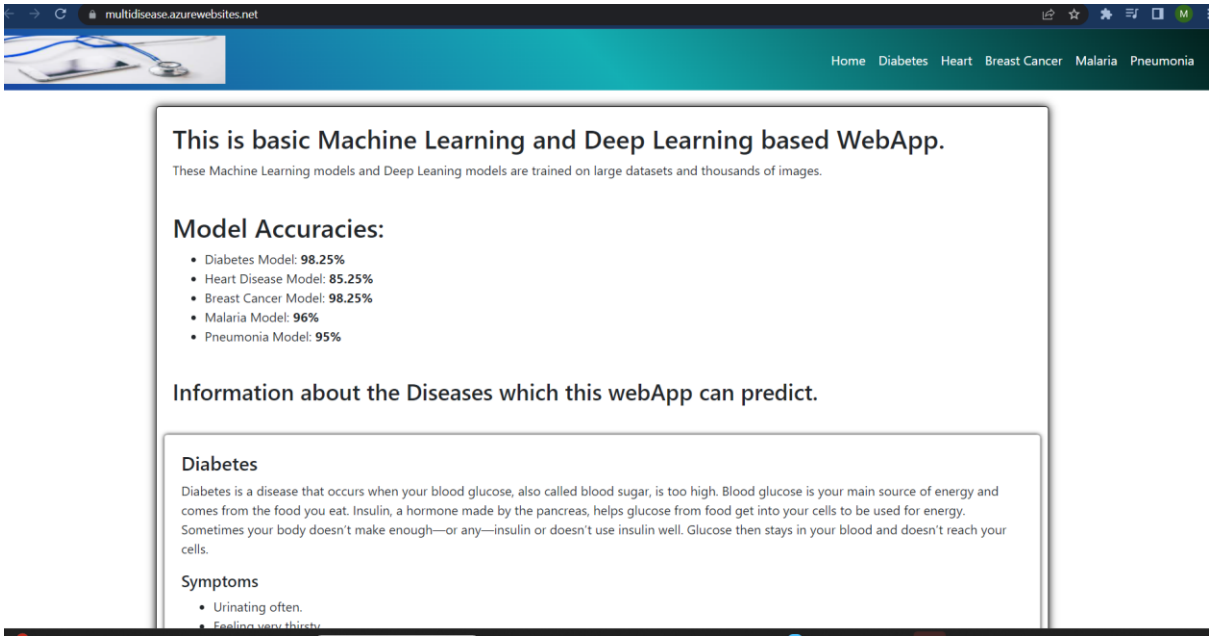
Code snippet of python :

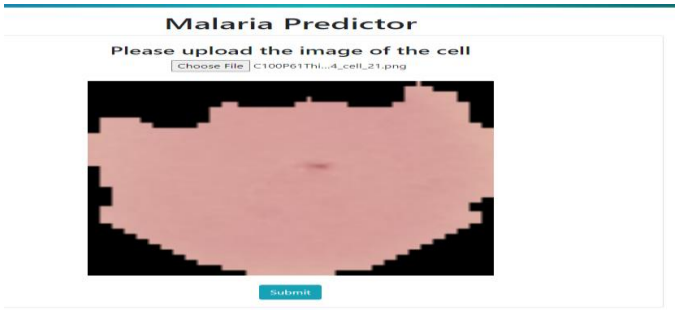
```
from flask import Flask, render_template, request, flash, redirect
import pickle
import numpy as np
from PIL import Image
from tensorflow.keras.models import load_model



app = Flask(__name__)

def predict(values, dic):
    if len(values) == 8:
        model = pickle.load(open('models/diabetes.pkl','rb'))
        values = np.asarray(values)
        return model.predict(values.reshape(1, -1))[0]

    elif len(values) == 26:
        model = pickle.load(open('models/breast_cancer.pkl','rb'))
        values = np.asarray(values)
        return model.predict(values.reshape(1, -1))[0]

    elif len(values) == 13:
        model = pickle.load(open('models/heart.pkl','rb'))
        values = np.asarray(values)
```

```
        return model.predict(values.reshape(1, -1))[0]


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

@app.route("/diabetes", methods=['GET', 'POST'])
def diabetesPage():
    return render_template('diabetes.html')

@app.route("/cancer", methods=['GET', 'POST'])
def cancerPage():
    return render_template('breast_cancer.html')


@app.route("/heart", methods=['GET', 'POST'])
def heartPage():
    return render_template('heart.html')


@app.route("/malaria", methods=['GET', 'POST'])
def malariaPage():
    return render_template('malaria.html')

@app.route("/pneumonia", methods=['GET', 'POST'])
def pneumoniaPage():
    return render_template('pneumonia.html')

@app.route("/predict", methods = ['POST', 'GET'])
def predictPage():
    try:
        if request.method == 'POST':
            to_predict_dict = request.form.to_dict()
            to_predict_list = list(map(float, list(to_predict_dict.values())))
            pred = predict(to_predict_list, to_predict_dict)
    except:
        message = "Please enter valid Data"
        return render_template("home.html", message = message)

    return render_template('predict.html', pred = pred)

@app.route("/malariapredict", methods = ['POST', 'GET'])
def malariapredictPage():
    if request.method == 'POST':
        try:
            if 'image' in request.files:
                img = Image.open(request.files['image'])
                img = img.resize((36,36))
```

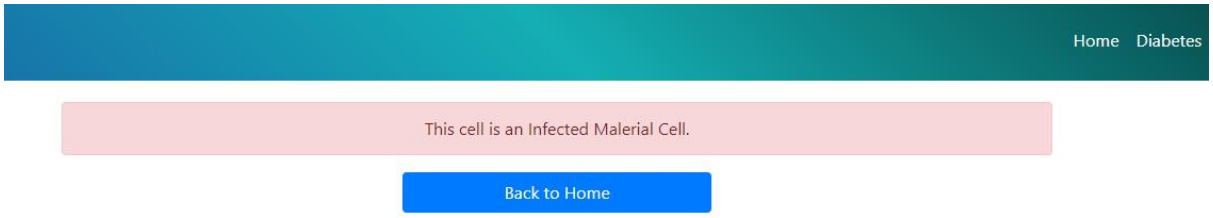
```
            img = np.asarray(img)
            img = img.reshape((1,36,36,3))
            img = img.astype(np.float64)
            model = load_model("models/malaria.h5")
            pred = np.argmax(model.predict(img)[0])
        except:
            message = "Please upload an Image"
            return render_template('malaria.html', message = message)
    return render_template('malaria_predict.html', pred = pred)

@app.route("/pneumoniapredict", methods = ['POST', 'GET'])
def pneumoniapredictPage():
    if request.method == 'POST':
        try:
            if 'image' in request.files:
                img = Image.open(request.files['image']).convert('L')
                img = img.resize((36,36))
                img = np.asarray(img)
                img = img.reshape((1,36,36,1))
                img = img / 255.0
                model = load_model("models/pneumonia.h5")
                pred = np.argmax(model.predict(img)[0])
        except:
            message = "Please upload an Image"
            return render_template('pneumonia.html', message = message)
    return render_template('pneumonia_predict.html', pred = pred)

if __name__ == '__main__':
    app.run(debug = True)
```

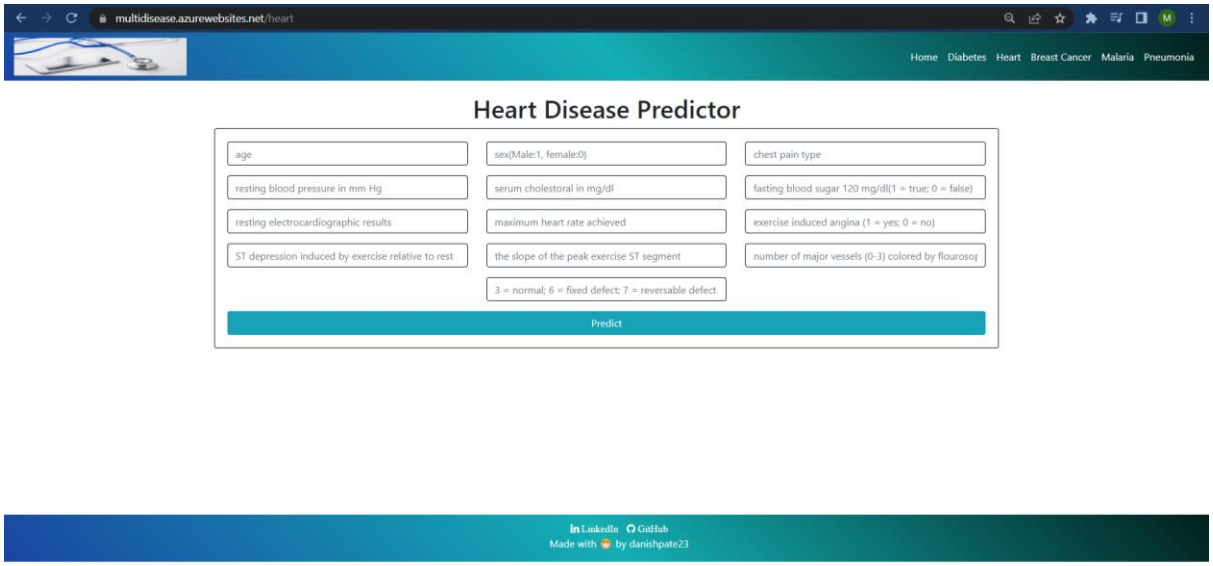
SCREENSHOTS :





After submit



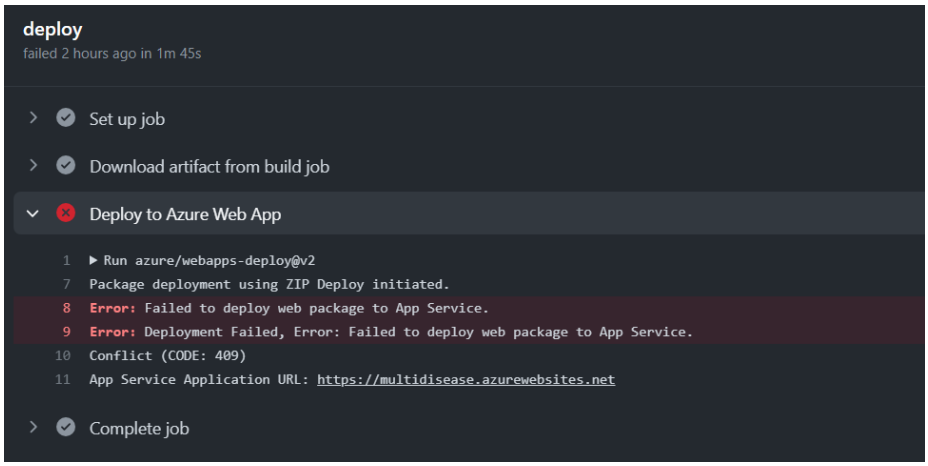


Challenges faced in GitHub Actions ZIP deployment and Python packages:

1. Deployment Failure: ZIP deployment in GitHub Actions may fail due to package structure, dependency conflicts, or version mismatches, requiring thorough troubleshooting to identify and resolve the issue.
2. Package Compatibility: Ensuring compatibility between Python packages can be challenging, with potential conflicts or dependencies on specific versions, requiring careful management and version control to avoid deployment errors.

3. Package Size Limitations: GitHub Actions has size limitations for deployment packages. Large package sizes can exceed these limits, leading to deployment failures. Managing package sizes, especially with large datasets or multiple libraries, becomes crucial.

4. Dependency Management: Managing dependencies and ensuring their availability during deployment can be challenging. Some packages may have additional dependencies or require specific system configurations, necessitating proper handling and provisioning in the deployment environment.



Business Benefits :

Improved Healthcare Access: The WebApp bridges the gap in healthcare access by providing individuals in rural areas with an accessible platform for disease detection and diagnosis, empowering them to take proactive measures for their health.

Early Disease Detection: By leveraging machine learning algorithms, the WebApp enables early detection of diseases such as pneumonia, malaria, diabetes, heart disease, and breast cancer, leading to timely medical intervention and potentially improved treatment outcomes.

Cost Savings: Timely disease detection and early intervention can potentially reduce long-term healthcare costs by preventing complications and minimizing the need for extensive and expensive treatments.

Increased Awareness and Education: The WebApp promotes health awareness and education by providing users with information about various diseases, risk factors, and preventive measures, empowering them to make informed decisions about their health.

Enhanced Agricultural Productivity: By addressing the healthcare needs of farmers, the WebApp contributes to improving agricultural productivity by ensuring that farmers are in good health, thereby reducing productivity losses due to illness.

References to Microsoft Documentation:

Azure App Service Documentation: https://docs.microsoft.com/azure/app-service/
Quickstart: Create a web app in Azure App Service: https://docs.microsoft.com/azure/app-service/quickstart-create-web-app
Azure App Service Deployment Center: https://docs.microsoft.com/azure/app-service/deploy/
Azure Virtual Machines (VM):

Azure Virtual Machines Documentation: https://docs.microsoft.com/azure/virtual-machines/
Quickstart: Create a Windows virtual machine in the Azure portal: https://docs.microsoft.com/azure/virtual-machines/windows/quick-create-portal
Azure Virtual Machines pricing: https://docs.microsoft.com/azure/virtual-machines/pricing
GitHub Actions:

GitHub Actions Documentation: https://docs.github.com/en/actions
Getting started with GitHub Actions: https://docs.github.com/en/actions/getting-started-with-github-actions