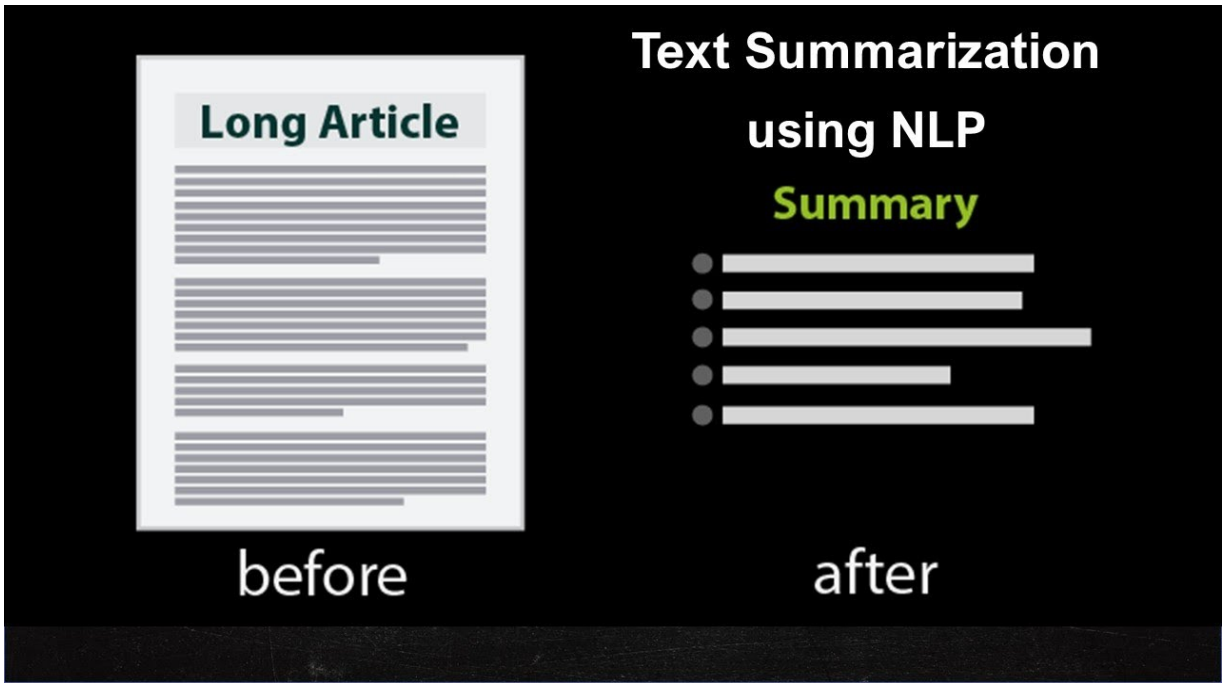
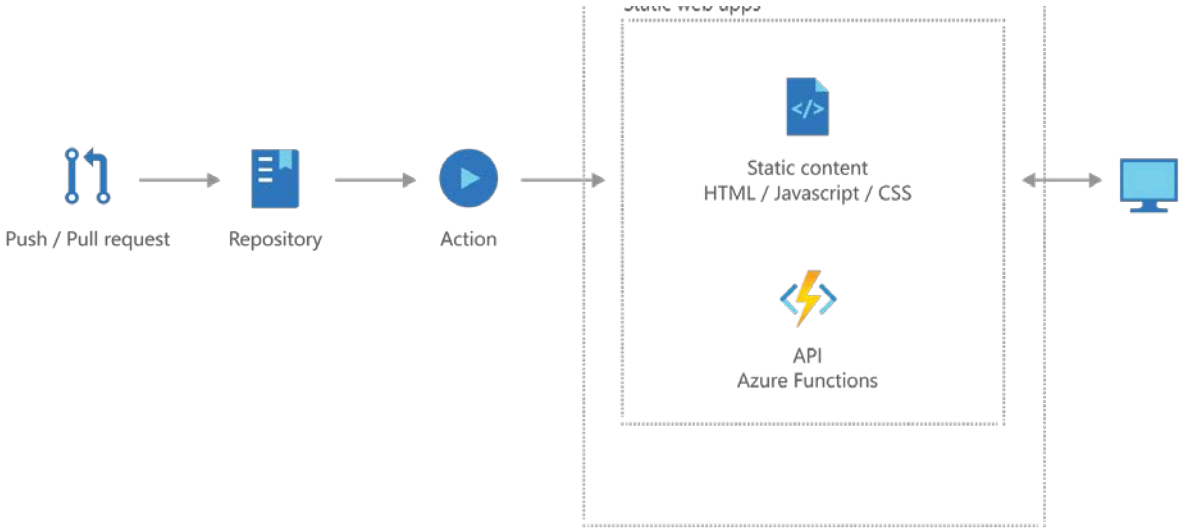
USING PYTHON AND FLASK DEPLOYED TEXT SUMMARIZER APP



Problem Statements

This article discuss about creating an application where a user can enter its large and massive amount of textual information and can get a short summary of its information. Most of the times, while searching through large data, user gets distracted and lose focus. Many times the data is unstructured and user needs to read thoroughly the whole blog/e-book/article and then reach conclusion about its effectiveness and get his required information. It is time inefficient task in this ever growing digital media world. In this post you will learn how to implement a solution using Python and Flask, and hosting it on Azure App Services. You will also learn to use Azure Cognitive service i.e. document summarization APIwhich uses natural language processing techniques.

Solution :



The algorithm is very simple. First you will have to enter the information. Then select the length of summary required. Then the application will parse through the content and using the “Hugging Face-Natural Language Processing Summarization API” it will convert the content into short summary and provide it to you.

Following are the packages used in this example.

Package Name	Description
Flask	For user interface and user interactions.
Hugging Face API	For transforming the content into the required length of summary using NLP.

```
Flask==2.0.2
requests==2.26.0
```

So you need to install the above packages. Here is the requirement.txt file.

You can run the `pip install -r requirements.txt`in your virtual environment. Once you install all the requirements, you can create the `app.py`file. You can find the `app.py`file in the implementation section. Visual studio code (VS Code) can be used for development.

```
"https://api-inference.huggingface.co/models/facebook/bart-large-cnn"
```

To use Hugging Face API you need to insert above link into your program.

Apart from this you will also need `index.html`file for creating the application's interface for deploying on the web along with the `style.css`file.

CODE FOR CREATING THE APPLICATION

```
import requests
from flask import Flask,render_template,url_for
from flask import request as req

app = Flask(__name__)
@app.route("/",methods=["GET","POST"])
def Index():
    return render_template("index.html")
@app.route("/Summarize",methods=["GET","POST"])
defSummarize():
if req.method== "POST":
        API_URL = "https://api-inference.huggingface.co/models/facebook/bart-large-cnn"
        headers = {"Authorization": f"Bearer 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"}
        data=req.form["data"]
        maxL=int(req.form["maxL"])
        minL=maxL//4
        def query(payload):
            Response = requests.post(API_URL, headers=headers, json=payload)
            return response.json()
        output = query({
            "inputs":data,
            "parameters":{"min_length":minL,"max_length":maxL},
        })[0]

        return render_template("index.html",result=output["summary_text"])
else:
        return render_template("index.html")
```

IMPLEMENTATION

You can use the Flask framework to show the user interface and interact with user inputs. The Hugging Face API is for transforming the larger content into short summarized form.

This implementation has one route with different HTTP methods. When a user browse the URL, the HTTP GET method gets invoked and returns`html.index`file. And when a user fills the information in the `input field`and selects the required size of summary and clicks on Summarize button, the HTTP POST route gets invoked. In backend, you will get the information in input field. Using the `Hugging Face API`, the information is parsed and next using `requests.post()`the content is summarized into smaller sentences. And finally, based on the required size of summary the summarized text is placed into json file and render the`index.html`file with the json file. You can run/debug the file using VS Code.

In the next section, you will publish the solution to Azure.

DEPLOYING TO AZURE

Flow of Application Deployment

To deploy the solution to Azure, first create a repository on Github containing all the files of your project. You will need this repository further. You will also need the `requirements.txt`file with `flask`and requests packages. You need to add the packages you installed to this.

```
autopep8==1.6.0
certifi==2021.10.8
charset-normalizer==2.0.7
click==8.0.3
colorama==0.4.4
Flask==2.0.2
idna==3.3
importlib-metadata==4.8.1
itsdangerous==2.0.1
Jinja2==3.0.2
MarkupSafe==2.0.1
pycodestyle==2.8.0
requests==2.26.0
toml==0.10.2
typing-extensions==3.10.0.2
urllib3==1.26.7
Werkzeug==2.0.2
zipp==3.6.0
```

Modify the `requirements.txt`file like the following.

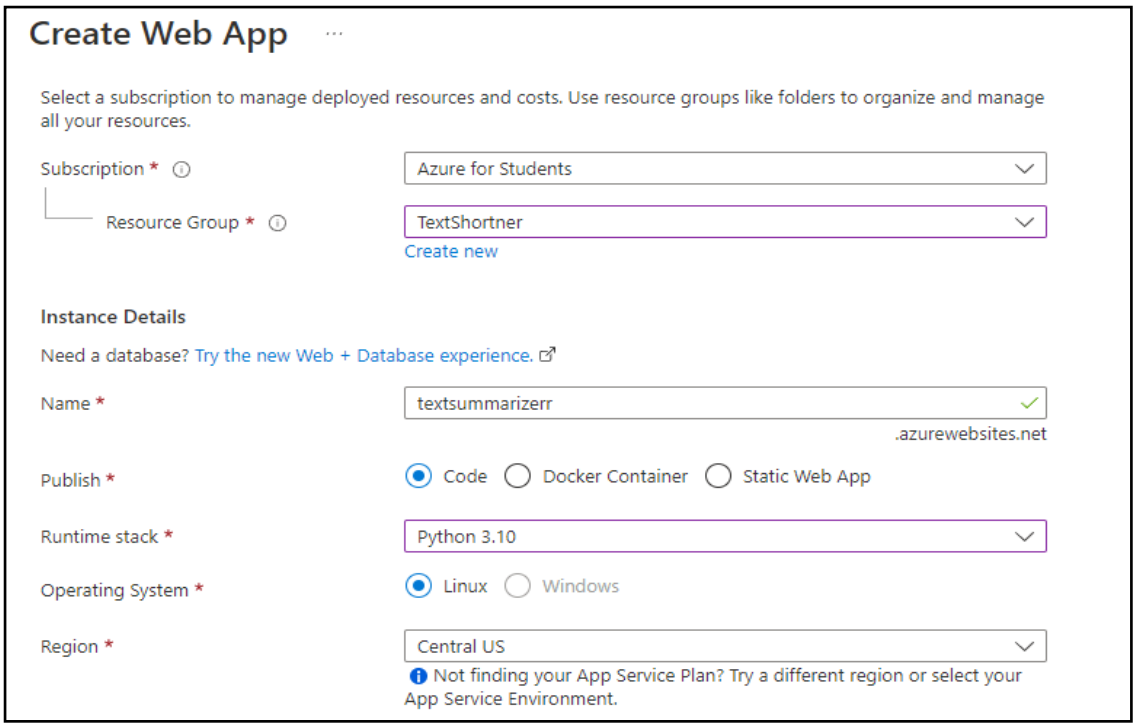
Then for deploying the application follow following steps.

Pre-requisites:

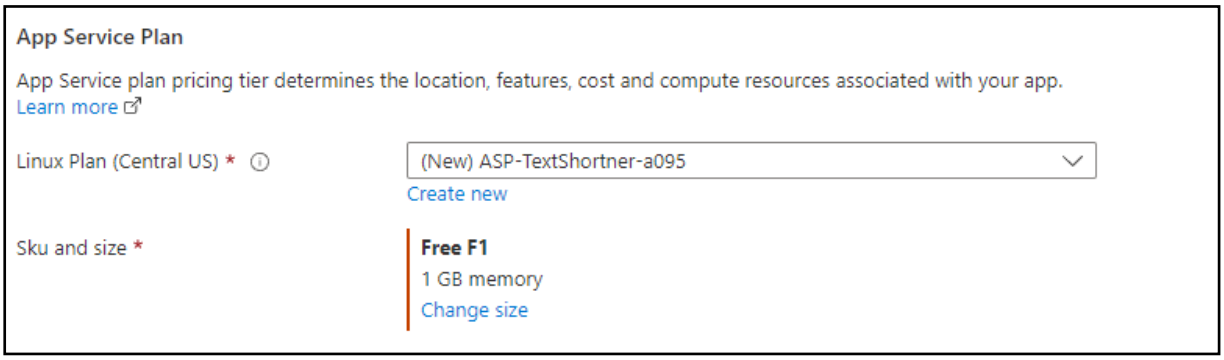
You need an Azure account with active Azure subscription.

Step 1: Create App Service

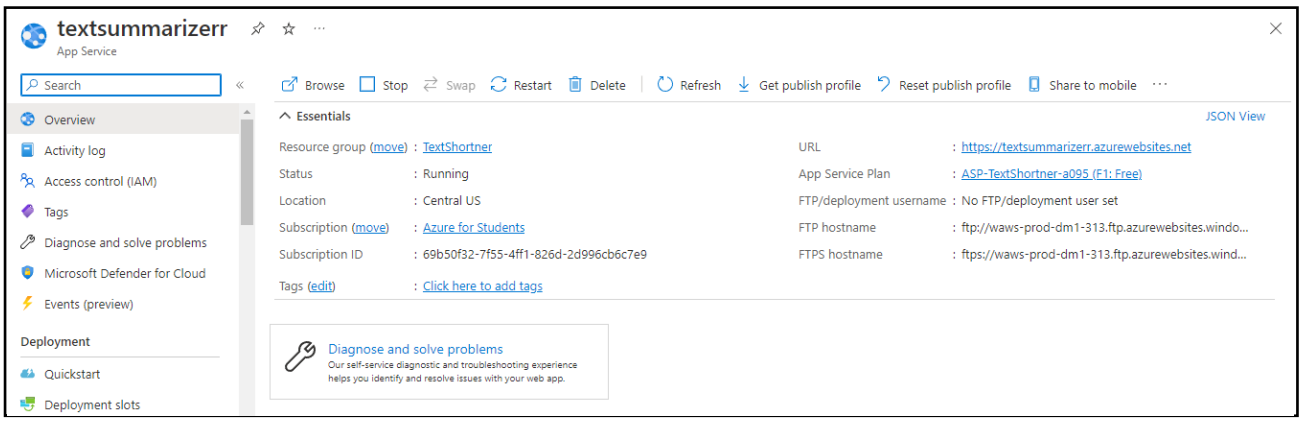
Login to azure portal and navigate to Azure App Service. Create new App Service. Configure the resource group name, the App Service name, select runtime as `Python 3.10`and select the region of your choice.



Select the Linux-based plan for App Service and let instance SKU be the F1 (Free) tier one, you will find that in the Dev/Test section.



Then click on Review and Create and create the Web App. It will take a few minutes for Web App to be ready. Once ready, note the resource group name, and the public endpoint URL of the Web App.



Step 2: Prepare your flask application to deploy

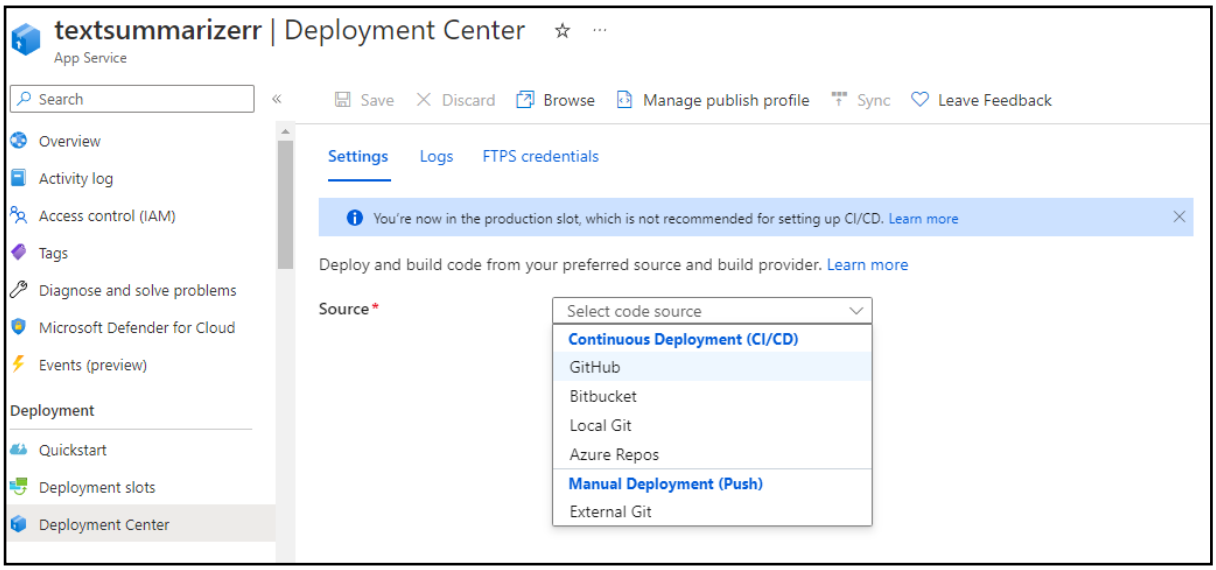
We are deploying a Flask-based application on the App Service. Name the application code as `app.py`.Make sure that the `requirements.txt`(which we already created) and the `app.py`files are in the same directory. Push code to the repository which we already created on Github.

Step 3: Link code to the application

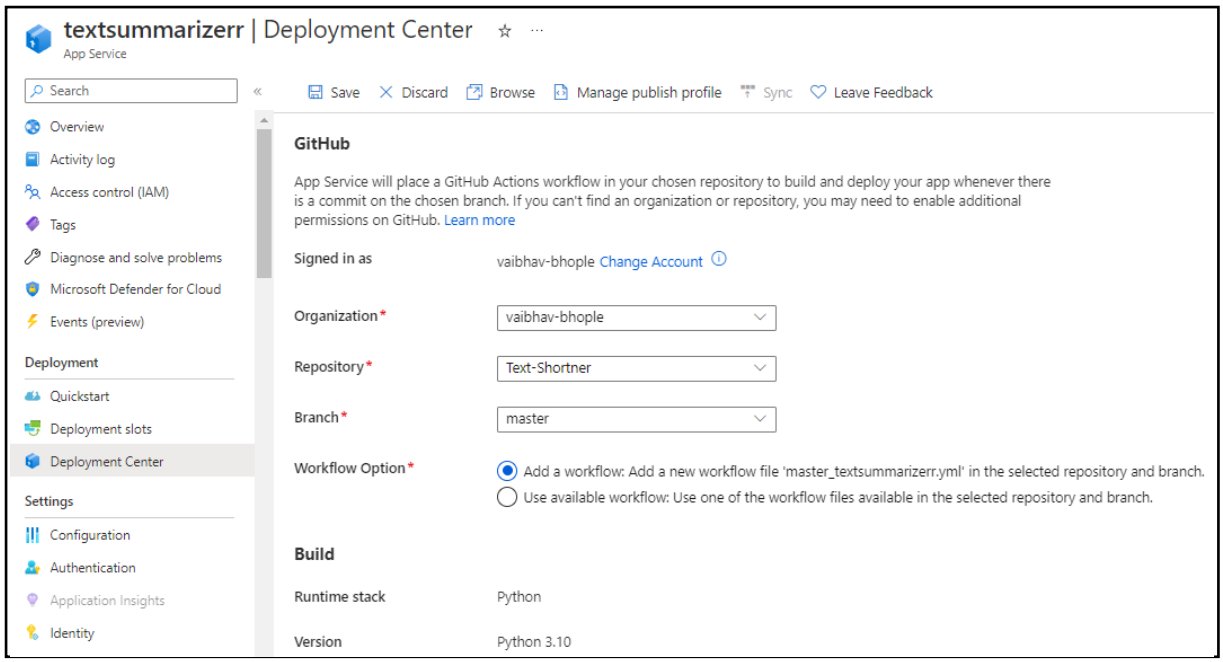
https://github.com/vaibhav-bhople/Text-Shortner

We are going to link the code to the application by linking the repository.

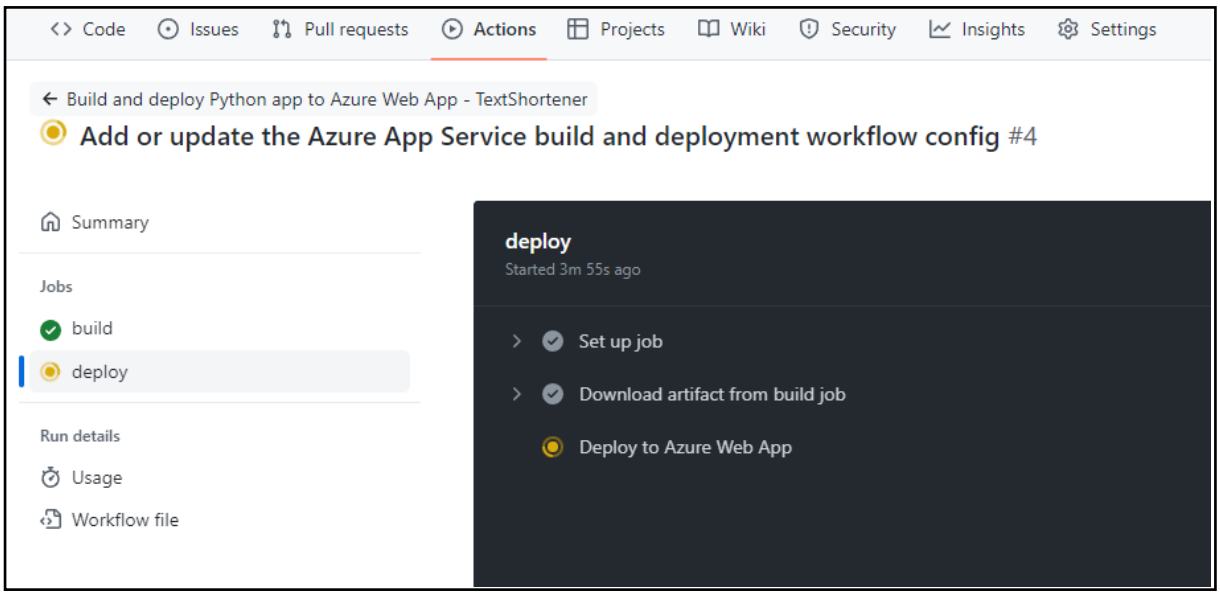
Go to Deployment Center and for source codeselect Github.



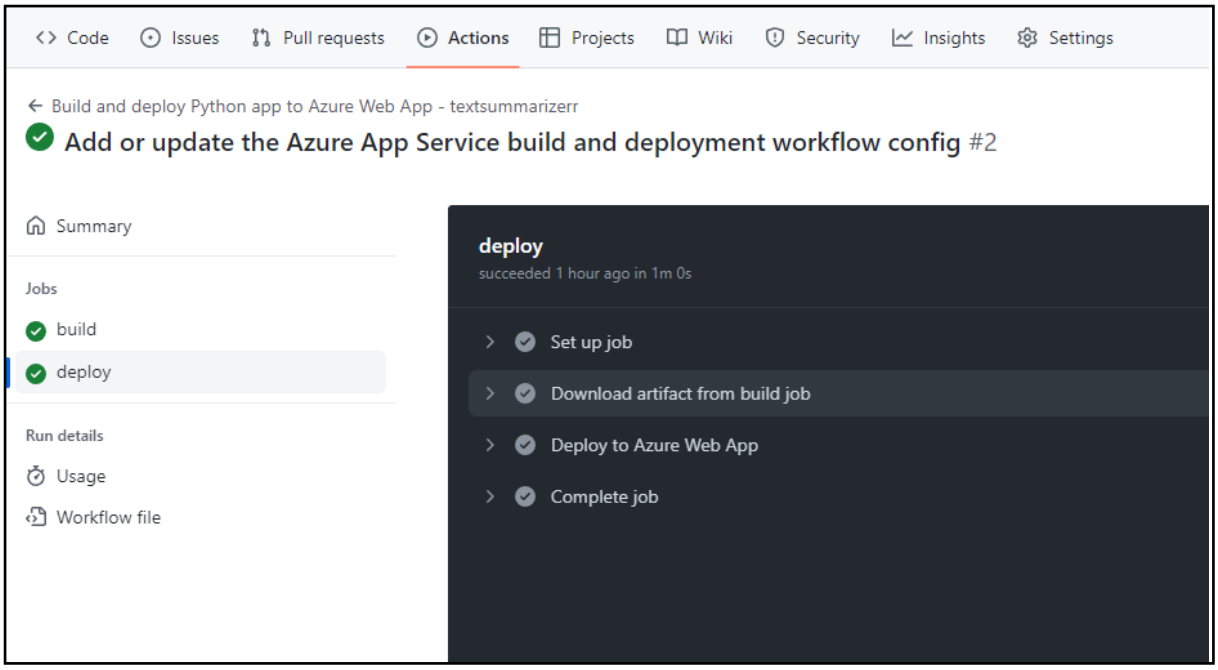
Then sign in with your Github account and configure organization name as your Github username and select your repository and branch as master. If you have a workflow file in your repository select available workflow else select add a workflow. You can see for build Runtime stack and version is already selected as Python.



Then click on save to set up deployment. Then the deployment workflow will begin. You can track the deployment progress in your repository in the Actions section.



It will take a couple of minutes to download the libraries from requirements and the deployment will be successful. Wait for the deployment to finish.



As soon as the workflow is successfully completed, your app is successfully deployed on Azure.

Check on : https://textsummarizerr.azurewebsites.net

Challenges Faced:

- Identifying the dependencies and creating the requirements.txt file.
- App compatibility issues.
- Took more time for deployment than usual.
- Identifying the errors while interruptions during deployment.
- Azure provides security to your application but for information security and higher security it is advisable to create and implement a virtual private network having end-to-end encryption.

Business Benefit

- Offers highly secure web apps development.
- Offers global scalability with high availability.
- Secure integration with any source codes.
- Services are charged on pay-as-you-go model, which helps saving a lot of money.

By Mr. VAIBHAV BHOPLE