# DEPLOYING PHONEBOOK APPLICATION IN MICROSOFT AZURE

## APPLICATION FOLDER STRUCTURE

This application is a simple contact manager built with React and TypeScript. It allows users to add, view, update, and delete their contacts. The application is styled using Bootstrap and custom CSS.

Here is a brief overview of the main components and files:

1. src/App.tsx:  
This is the main component of the application. It renders the `ContactForm` and `ContactList` components and manages the state of the application.

2. src/components/ContactForm.tsx:   
This component renders a form that allows users to add a new contact or update an existing one. It takes in the dispatch function from the `App` component to dispatch actions to the reducer.

3. src/components/ContactList.tsx:

This component renders a list of all contacts. It also takes in the dispatch function from the `App` component to dispatch actions to the reducer.

4. src/components/EditModal.tsx:  
This component renders a modal that allows users to edit a contact's details.

5. src/reducers/contactsReducer.ts.:   
This file contains the reducer function for the application. The reducer function takes in the current state and an action, and returns the new state based on the action type.

6. src/types.ts:  
This file contains the TypeScript interfaces and types used in the application.

7. package.json:  
 This file contains the list of dependencies and scripts for the application. You can start the application by running `npm run dev` and build it for production by running `npm run build`.

8. github/workflows/main\_phonebook-app.yml:  
This file contains the configuration for the GitHub Actions workflow. The workflow is set up to build and deploy the application to Azure Web App whenever changes are pushed to the master branch.

9. README.md:  
This file contains instructions on how to install, use, and contribute to the application.

10. tsconfig.json:  
This file contains the TypeScript compiler options for the application.

## MAIN FUNCTIONS USED

Based on the provided workspace structure and the active selection, here are the main functions used in the application:

1. `main` function in main.tsx:  
This function is the entry point of the application. It renders the `App` component into the root element of the HTML document.

2. `dispatch` function in ContactItem.tsx:  
This function is also passed as a prop to the `ContactItem` component. It is used to dispatch actions to the reducer function in contactsReducer.ts. The actions can be of different types, such as adding a new contact, updating an existing contact, or deleting a contact.

3. `render` function in App.tsx:  
This function is responsible for rendering the main layout of the application. It renders the `ContactForm` and `ContactList` components and passes the necessary props to them.

4. `handleEdit` function in ContactItem.tsx:  
 This function is passed as a prop to the `ContactItem` component. It is used to handle the editing of a contact. When the edit icon is clicked, this function is called with the id of the contact to be edited.

5. `reducer` function in contactsReducer.ts:  
This function is used to manage the state of the application. It takes the current state and an action as arguments, and returns the new state based on the action type.

## HOW TO USE THE APPLICATION

To use the application, you can add a new contact by clicking on the "Add Contact" button and filling out the form. To view the details of a contact, simply click on the contact's name in the list. To edit a contact's details, click on the edit icon next to the contact's name and make the necessary changes in the form. To delete a contact, click on the delete icon next to the contact's name and confirm the action when prompted. Any changes made to the contacts are automatically saved to the local storage.

HOW TO BUILD THE APPLICATION  
  
To install all the node packages run `npm install` in the terminal.  
To start the application, run `npm run dev` in the terminal.  
  
  
HOW TO DEPLOY APP INTO MICROSOFT AZURE  
  
You should have the following prerequisites before we start:

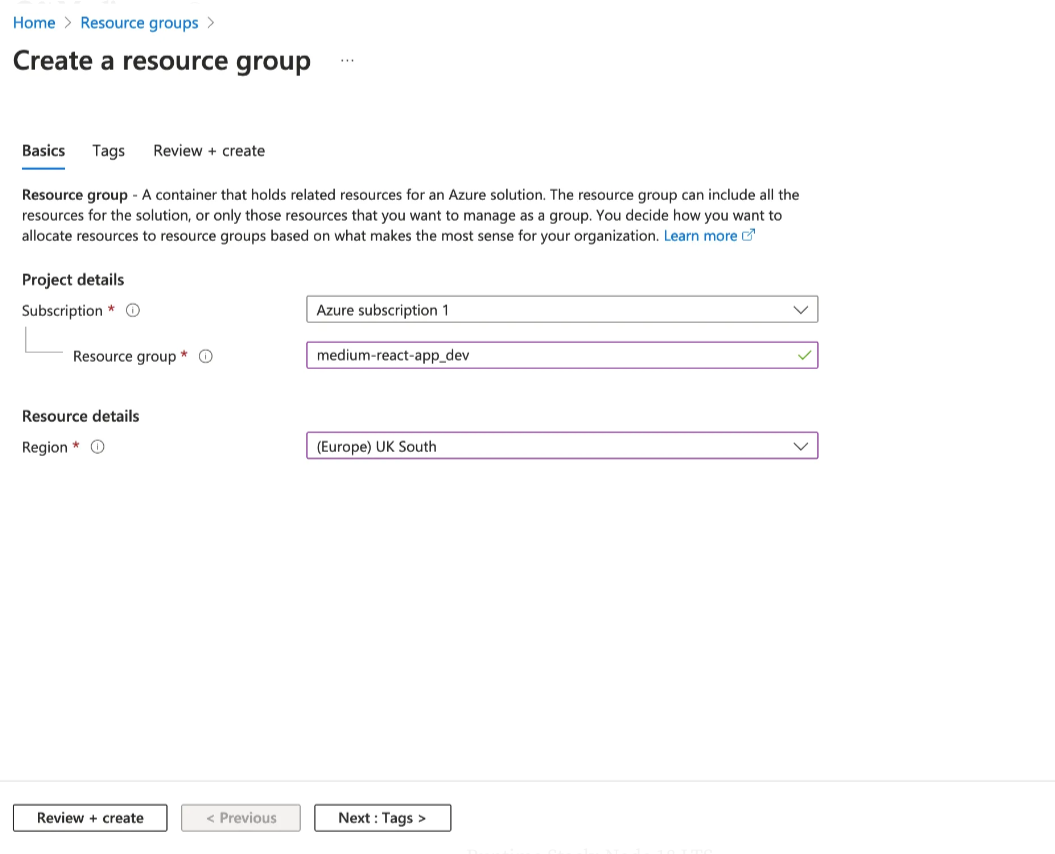
* A Microsoft Azure account with active subscription
* Latest Node LTS (v18.16.1 ATTOW) installed in your system
* A GitHub account

We are deploying our application in Microsoft azure using Azure App Service. Azure App Service is Azure’s Platform as a Service (PaaS) for hosting web apps, RESTful APIs, and mobile backends. It also comes with built-in DevOps capabilities, including continuous deployment using GitHub Actions.

# Step by Step Guide

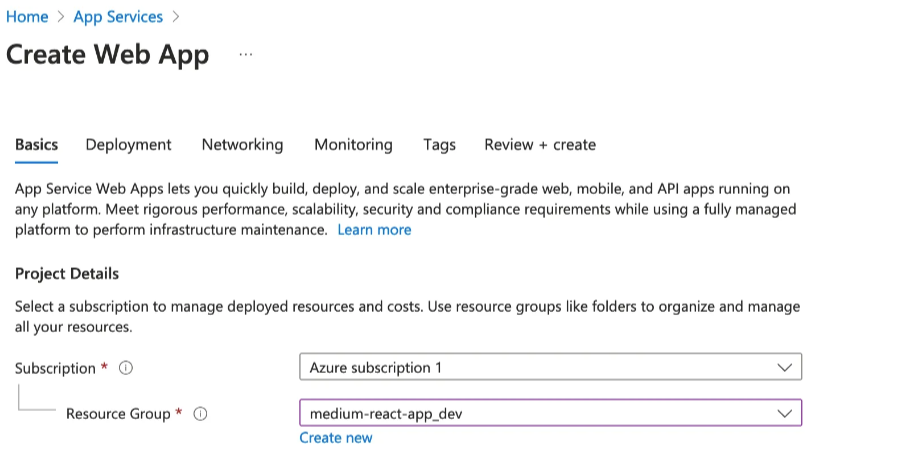
## Creating an app service on Azure using the Azure Portal

1. **Go to Home — Resource Group — Create.**

Create a resource group named {name\_of\_your\_app}\_dev in your region, and click review + create. This will be our development resource group. If needed, we can also create testing and production groups later.  
  


**2. Go to Home — App Services — Create — Web App**

Select your subscription, and the same Resource Group that we just created:



Then fill in instance details with the following:

Name: {name\_of\_your\_app}

Publish: code

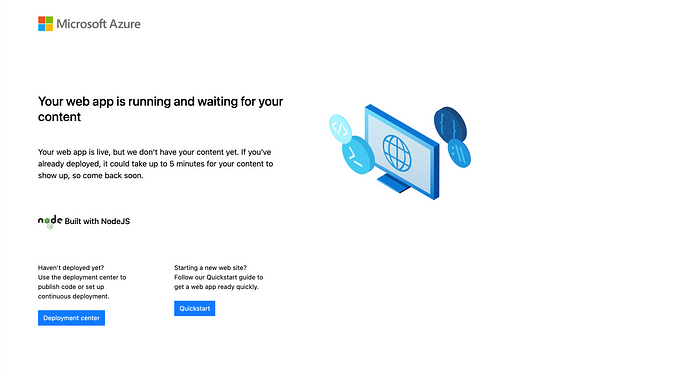
Runtime Stack: Node 18 LTS

Operating System: Linux or Windows depending on your own preference. I’m going with Linux here.

Region: the same as your resource group

Pricing plan: select your own pricing plan. For personal projects, the free F1 plan will be more than sufficient.

After all the fields have been filled in, we can click next to configure deployment.  
  
After the deployment is complete, which should only take a few minutes, go to **Home** — **App Services**, and we should now see our new App Service and will be able to check it out.  
  
Click on the Browse button, or go to this url: https://{name\_of\_your\_app}.azurewebsites.net, and we should see the following page. It is a generic troubleshooting page from Azure indicating that they haven’t received any content for our web app.



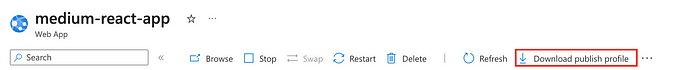
**3**. let’s now deploy our project and set up continuous deployment on Azure.  
  
In our project root directory, create a **.github/workflows/** folder and add the following workflow file named **main\_{name\_of\_your\_app}.yml**:

# Docs for the Azure Web Apps Deploy action: https://github.com/Azure/webapps-deploy  
# More GitHub Actions for Azure: https://github.com/Azure/actions  
  
name: Build and deploy Node.js app to Azure Web App - {name\_of\_your\_app}  
  
# Run the workflow whenever change is pushed into the main branch  
on:  
 push:  
 branches:  
 - main  
 workflow\_dispatch:  
  
# Optional: environment variables can be injected here  
# You can add these variables in your repository Settings - Secrets and variables - Actions - variables  
env:  
 VITE\_ENV\_VAR1: ${{ env.VITE\_ENV\_VAR1 }}  
 VITE\_ENV\_VAR2: ${{ env.VITE\_ENV\_VAR2 }}  
  
# There are two jobs in this workflow - build and deploy  
jobs:  
 # Builds the application for production.   
 # You should configure this environment as close as possible to the production environment, e.g. using Ubuntu for Linux apps, using the same Node version  
 build:  
 runs-on: ubuntu-latest  
  
 steps:  
 - uses: actions/checkout@v3  
  
 - name: Set up Node.js version  
 uses: actions/setup-node@v3  
 with:  
 node-version: '18.x'  
  
 - name: Install dependencies, build, and test  
 run: |  
 npm install  
 npm run build --if-present  
 npm run test --if-present  
 # Other scripts if configured should be added here, such as linting, formating  
  
 - name: Upload artifact for deployment job  
 uses: actions/upload-artifact@v3  
 with:  
 name: node-app  
 path: dist # This should be your distribution path, not the root path  
   
 # Deploy the application to Azure  
 deploy:  
 runs-on: ubuntu-latest  
 needs: build  
 environment:  
 name: 'Production'  
 url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}  
  
 steps:  
 - name: Download artifact from build job  
 uses: actions/download-artifact@v3  
 with:  
 name: node-app  
  
 - name: 'Deploy to Azure Web App'  
 id: deploy-to-webapp  
 uses: azure/webapps-deploy@v2  
 with:  
 app-name: {name\_of\_your\_app}  
 slot-name: 'Production'  
 publish-profile: ${{ secrets.AZUREAPPSERVICE\_PUBLISHPROFILE }} # Publish profile should always be stored as secrets in your repository Settings - Secrets and variables - Actions - secrets  
 package: .

Before committing this file to the main branch, we will need to configure secrets and environment variables (if any). Secrets are basically environment variables that are confidential. Once a secret is created, you can delete or overwrite it, but you will not be able to view it.

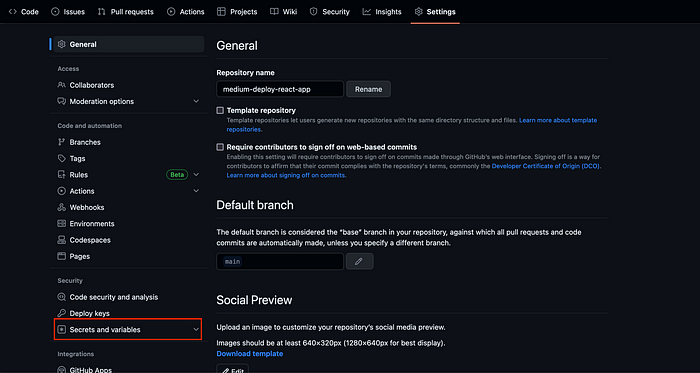
The Azure Publish Profile holds confidential information that is required for deploying to Azure app service. It is sensitive data, so we will need to store it as a secret, which can then be used repeatedly in the workflow.

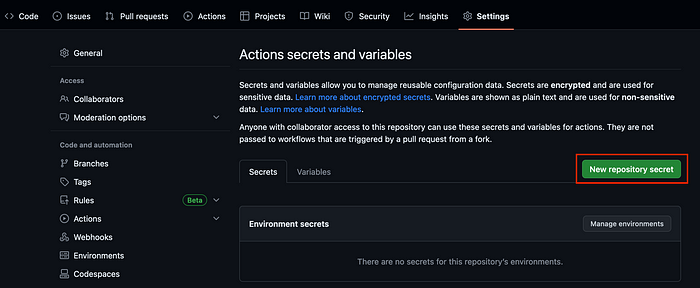
To get the publish profile, we can go to the **Overview** page of our app service, and click **Download Publish profile**.



Copy and paste the content of the downloaded file, as it will be used in the next step as the value for our GitHub secret. We can now leave the Azure Portal and navigate back to the GitHub project page.

We can add secrets and variables in Settings — Secrets and variables — Actions — secrets.





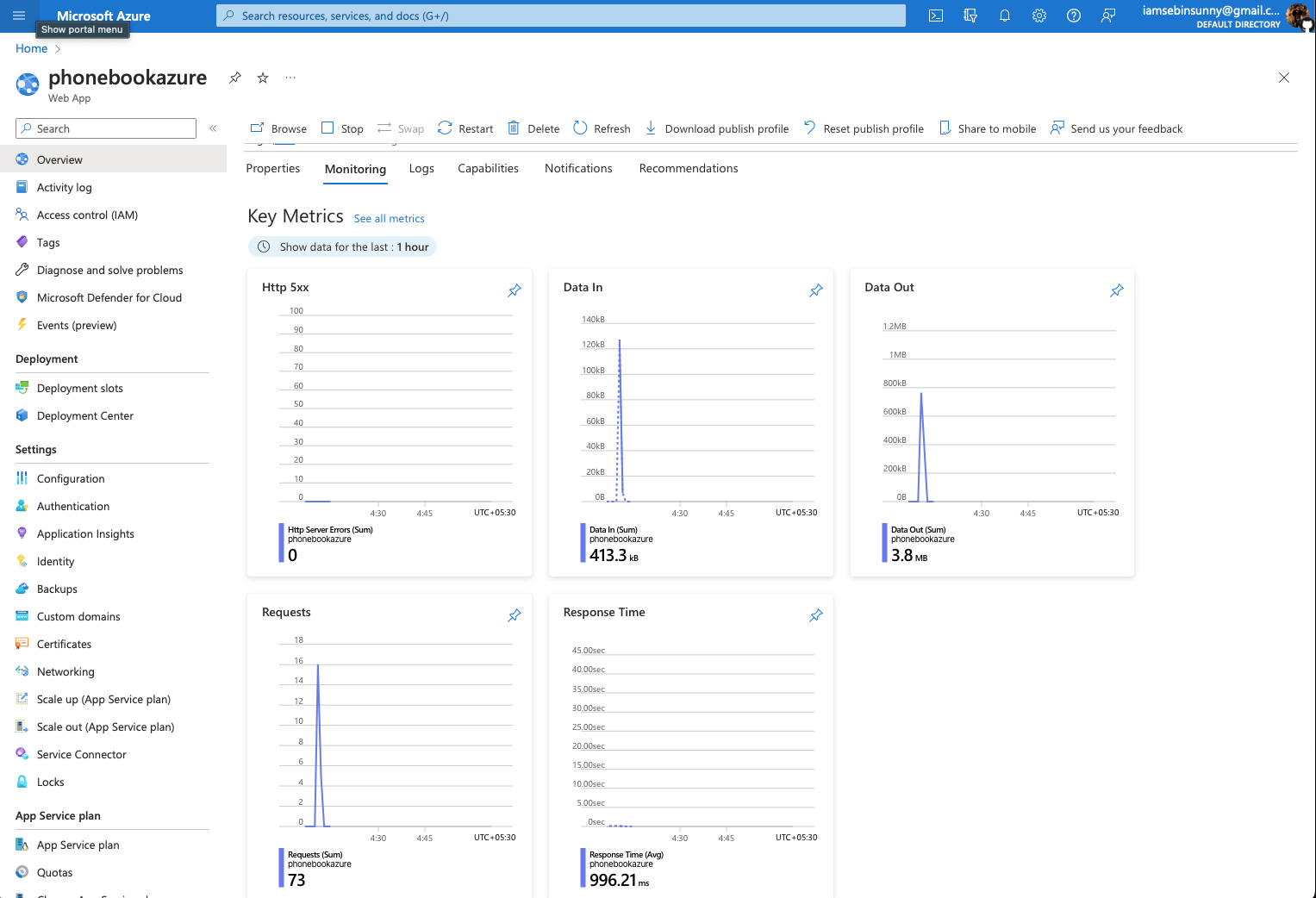
We will then create the following secret:

*AZUREAPPSERVICE\_PUBLISHPROFILE={your\_publish\_profile\_content}*

Now that all the setup is finished, if we push the workflow file to the main branch, we should see a workflow run under GitHub’s Action tab.

If the workflow has been successfully run, that means our React app has now been deployed to Azure.

HOW TO MONITOR OUR APPLICATION  
  
Go to resource group which was created and head to overview section.  
Under overview you can see monitoring part. Here we will get to know all the metrics related to our website.

  
  
  
  
GITHUB REPO OUR THE APPLICATION  
  
https://github.com/iamsebinsunny/phonebook-app/