**Front End Installation**

Operating System : Ubuntu

Technology : ReactJS

Running Web App using Development server

Steps:

1. Clone the github repo containing the front app
2. Run the command npm install then it will install the required packages for running the application
3. After the installation we can able to run the Web application in Development server using the command npm start

Running Web App Production Build In a web Server(Nginx)

Steps:

1. Clone the github repo containing the front app
2. Run the command npm install then it will install the required packages for running the application
3. To create the production build we need to run the command npm run build after the successful execution of the command it will create a folder called build it contain all the build files
4. Upload all the build files to nginx website deployment file location
5. Make changes to the web server configuration

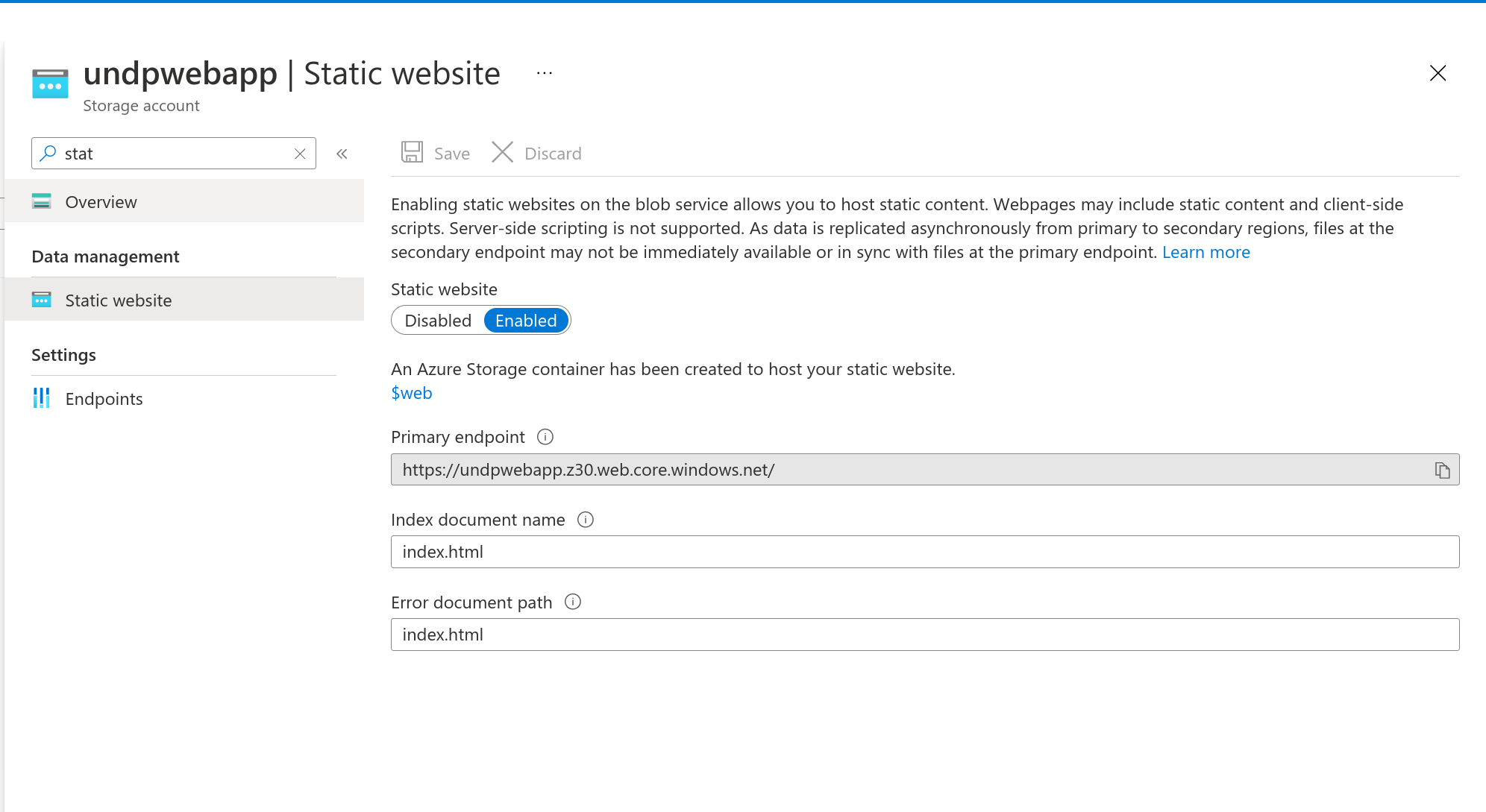
Running Web App Production Build in Azure Blob

Steps:

1. Clone the github repo containing the front app
2. Run the command npm install then it will install the required packages for running the application
3. To create the production build we need to run the command npm run build after the successful execution of the command it will create a folder called build it contain all the build files
4. To deploy react production build in Azure we need to create a storage account in Azure
5. After the successful Deployment of the storage account Goto static website menu and enable static website option and fill index document name as index.html and leave error document path as empty its optional

After saving this it will provide us a primary endpoint

Screenshot of the same is given below we can use the primary endpoint to test our react production build deployed in the storage account is working or not after the completion of step 6



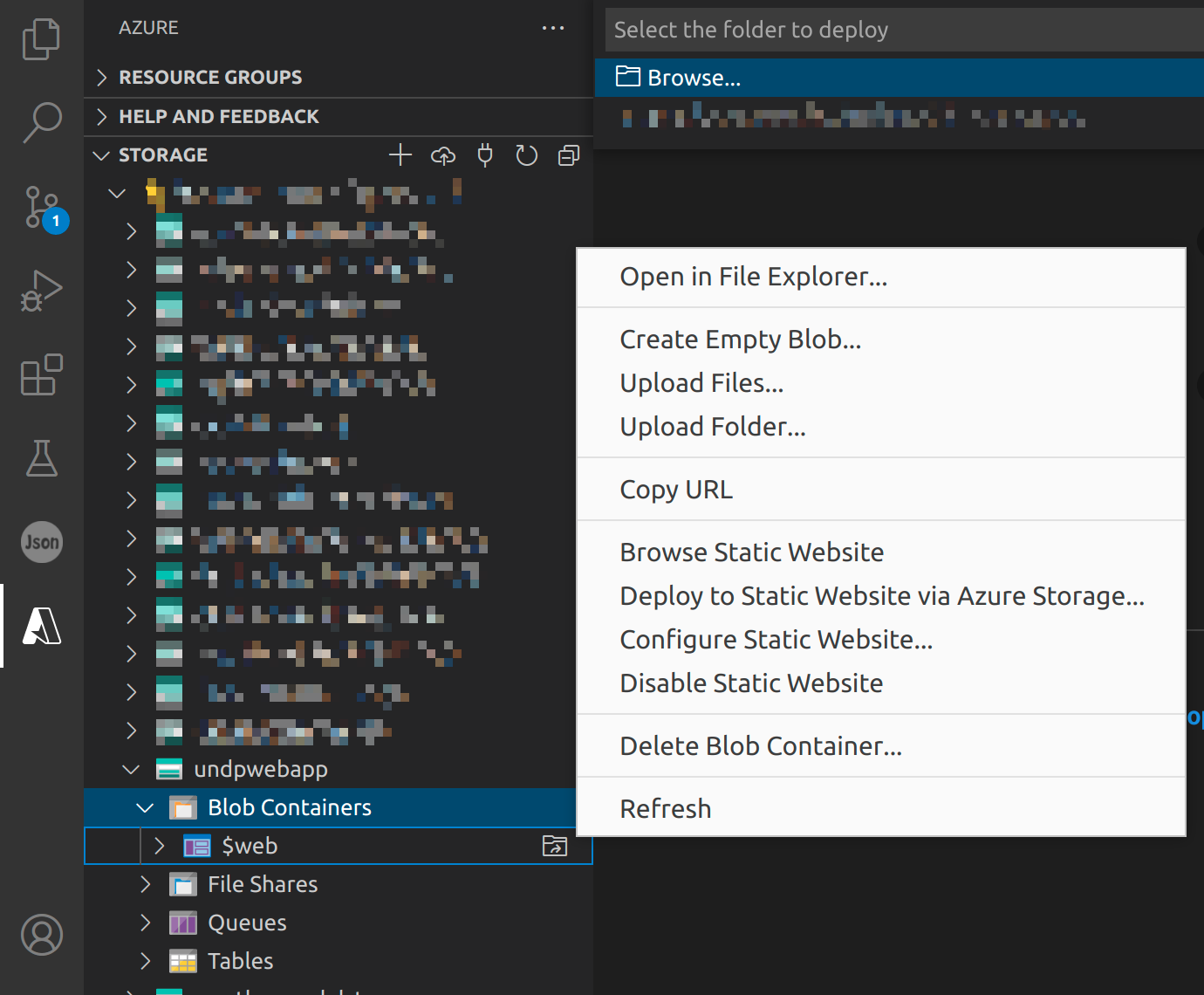
1. After the completion of Static website enabling section it will create two containers called $logs and we need to upload the build files created in step 3

We can upload Build files to $web container using multiple ways Azure storage explorer, Visual studio extension Azure storage by Microsoft

Steps followed to upload build files to $web using Azure storage is

* Goto azure storage extension on visual studio code
* Sign In using Azure credentials
* Expand storage account we have created for web app deployment
* Under the Blob Container menu we can able to see $web container right click on that choose option Deploy to static website via azure storage and browse the build folder and it will complete the deployment

After completing the above steps we can test the deployment using the primary endpoint



1. Create CDN profile for the front end

Create cdn profiles on Home->CDN profiles menu.

after successful deployment of cdn profile create cdn endpoint at the end point creation menu specify name as any meaningful name . origin type : Storage static website origin hostname : the hostname generated by the url when completing the first step

WE CAN ADD CUSTOM DOMAIN FOR OUR CDN HERE

1. Create Web application Firewall Policies (WAF)

Goto home -> Web application Firewall Policies (WAF) menu

Click create button

On the basic tab under the project details section select AzureCDN

Under the Instance details section select Policy mode select Prevention

Add necessary custom rules under Custom rules tab

Finally associate cdn end point at the Association tab and create the WAF policy

**Backend Installation**

Operating System: Ubuntu

Technology: Python

Database : PostgreSQL with postgis extension

Running Backend Server using uvicorn (Development server)

Steps:

1. Clone Github Repo containing Backend API
2. Install all required packages using the command pip install -r requirements.txt
3. To run all database migrations run alembic upgrade head

It will create all the necessary tables

1. Finally we can run the uvicorn development server using the command python main.py it will start a uvicorn development server [**http://localhost:5004**](http://localhost:5004)

**Running Backend server using gunicorn systemmd managed unit service and Caddy**

Steps:

1. Clone Github Repo containing Backend API
2. Create conda virtual environment using the command conda create -n environmentname python=3
3. Activate the conda virtual environment using the command conda activate envname
4. Install all the required packages using pip install -r requirements.txt
5. Change User,Group,WorkingDirectory,Environment in the gunicorn.service file in the repo
6. Create a gunicorn service by running

sudo nano /etc/systemd/system/gunicorn.service

1. Register the unit file gunicorn.service with Systemd by executing the following commands.

sudo systemctl daemon-reload

sudo systemctl enable gunicorn.service

sudo systemctl start gunicorn.service

The systemctl enable command above will add our gunicorn service to resume running when the VM reboots.

The systemctl start command will quickly start the gunicorn service and invokes the ExecStart command.

To check the status of our gunicorn.service at any point of time, run the following command.

sudo systemctl status gunicorn.service

1. Install caddy 2 web server

We can install caddy web server using the following command

echo "deb [trusted=yes] https://apt.fury.io/caddy/ /" | sudo tee -a /etc/apt/sources.list.d/caddy-fury.list

sudo apt update

sudo apt install -y caddy

We can check the caddy server status by running

systemctl status caddy

1. Now we will configure our Caddy 2 Web server to serve the FastAPI app running on port 8000 via a reverse proxy. To do so, lets edit the /etc/caddy/Caddyfile by running the following command.

sudo nano /etc/caddy/Caddyfile

Replace the contents of the Caddyfile and it should look like below

:80

reverse\_proxy 0.0.0.0:8000

Restart the caddy server by running the following command

sudo systemctl restart caddy