DOMAIN SETUP-

- domain provider- GoDaddy- we set up the domain by creating an account on GoDaddy and purchasing a domain name which is available

-DNS (domain name system), we use Cloudflare for this which is used for the following-

1. DNS Management- Handling DNS records (A, CNAME, etc)

2. DDoS protection- Helps protect our server from denial-of-service attacks

3. SSL security- provides basic HTTPS (TLS/SSL certificate) encryption

after this we need to set up Hostlinger which gives us a VPS and using its terminal fastpanel is installed which helps in setting up domains, managing SSL(HTTPS), changing the site webpage when someone clicks it, the backend

so, we purchased **Hostlinger VPS** to get full root access.

Installed **Fastpanel** (web-based control panel)

pointed our GoDaddy domain (thestockmood.com) to the VPS IP using A record.

we used to change the GoDaddy nameservers with the Cloudflare nameservers

configured SSL (HTTPS) using Fastpanel- SSL tab-

Logged into FastPanel (via browser using your VPS IP and FastPanel port — usually https://your_ip:8888).

Went to "Websites" tab in FastPanel.

Clicked on your domain:

thestockmood.com → opened its configuration panel.

from the main page of the webpage inside fastpanel went to SSL certifications

Selected “Free SSL by Let’s Encrypt” from the available options.

Made sure the following checkboxes were selected:

✅ Use SSL

✅ Redirect all HTTP to HTTPS (force HTTPS)

✅ Auto-renew (so SSL won’t expire)

Clicked “Install” — FastPanel fetched and installed the certificate from Let's Encrypt.

Confirmed HTTPS by visiting:

🔗 https://thestockmood.com

And checking for the padlock icon in the browser.

because of this-

All traffic is now encrypted (green padlock).

Your site is secure by default and trusted by browsers.

CLOUDFARE'S ROLE-

Cloudflare's Role in the StockMood Deployment

Primary Purpose: Cloudflare acts as a reverse proxy, DNS manager, and security/performance layer sitting between users and your actual server.

How Cloudflare Fits into Your Setup:

1. **Domain → Cloudflare DNS**

Your domain `thestockmood.com` was registered on GoDaddy.

Instead of using GoDaddy's default DNS, we pointed your domain's nameservers to Cloudflare (via GoDaddy settings).

This gave Cloudflare control over your domain's DNS — allowing us to manage `A`, `CNAME`, and other records there.

2. **DNS Records in Cloudflare**

We set an `A record`:

Name: @

Type: A

Value: YOUR_SERVER_IP

Proxy status: Proxied (orange cloud)

This means all traffic to `thestockmood.com` is routed **through Cloudflare**.

3. Reverse Proxy

Cloudflare becomes the middleman: User 🔁 → Cloudflare 🔁 → Your VPS (Hostinger)
This helps hide your real IP, control traffic, and filter malicious requests. Used fastpanel to do reverse proxy inside settings so that when the website first loads it goes to that link

4. CDN (Content Delivery Network)

Cloudflare caches static assets (images, JS, CSS) on global servers.

Improves page load speed for users worldwide.

5. Security

Cloudflare protects our VPS from:

DDoS attacks

Bad bots

Spam traffic

can enable a Web Application Firewall (WAF) if needed.

Why We Used Cloudflare:

Feature	Benefit
DNS Management	Easy and fast updates via Cloudflare UI
IP Masking	Keeps your VPS IP hidden
DDoS Protection	Blocks unwanted traffic
Global CDN	Faster load times
SSL Support	Works with Let's Encrypt
Analytics	Shows visitor traffic & threat data

Final Setup Flow:

User's Browser

↓

thestockmood.com DNS → Cloudflare (DNS + Proxy + CDN + Security)

↓

Cloudflare forwards request → Hostinger VPS (with Docker + FastPanel + SSL)

↓

Your Website (Vue frontend + Flask API + R API)

next we SSD'd into the server using ssh root@your_vps_ip

installed docker inside the server using the following commands-

apt update

apt install docker.io -y

then we installed the docker compose plugin-

apt install docker-compose -y

verified docker is working-

docker --version

docker compose version

earlier we did a project transfer, zipped the folder and transferred the file to the server using the following command-

scp StockMood.zip root@your_vps_ip:/root

then unzip and build-

unzip StockMood.zip

cd StockMood

docker compose down --volumes --remove-orphans

docker system prune -a

NGINX Domain Routing (via FastPanel)

Inside FastPanel, linked the domain thestockmood.com to the /frontend/dist output folder (served by NGINX).

Ensured reverse proxy routes to:

/api/ → Flask backend

/rapi/ or specific port → R API (if separate)

SSL enabled via FastPanel.

now the issue was that every time I needed to make a change I must transfer the files again on the server and delete the previous files including docker containers and images, I had to build the SQL database again and again. that's why GitHub

GitHub- a cloud-based git repository host, A platform to store, track, and collaborate on code, also supports automation like CI/CD, testing, and deploys (this is exactly what we needed).

It works through 3 components:

1. **GitHub Repository** – Your source code lives here

2. **GitHub Actions Workflow** – A `. yml` file that runs automated tasks like deployment

3. **SSH Key Authentication** – Secure access from GitHub to your server

How You Connected GitHub to the Server

1: You Created an SSH Key on Your PC

Inside bash- ssh-keygen -t rsa -b 4096 -C "contactstockmood@gmail.com"

This generated:

A private key: `C:\Users\madha\.ssh\id_rsa`

A **public key: `C:\Users\madha\.ssh\id_rsa.pub`

2: You Added the Public Key to Hostinger VPS

went to the Hostinger VPS dashboard

Clicked on SSH Keys

Pasted the public key from `id_rsa.pub`

This tells Hostinger: “Anyone with this private key can SSH into the server.”

3: You Added the Private Key to GitHub

went to our GitHub repo → **Settings > Secrets > Actions** and added:

Secret Name	Purpose
`VPS_HOST`	IP address of your server (`69.62.74.251`)
`VPS_USER`	Server username (`root`)
`VPS_SSH_KEY`	The **private key** from `id_rsa`

This gives GitHub permission to SSH into our server as `root`.

Step 4: You Created a GitHub Actions Workflow File

You added this file in your repo:

`.github/workflows/deploy.yml`

Inside deploy.yml-

```
- name: Copy Files to VPS
  uses: appleboy/scp-action@master
  with:
    host: ${{ secrets.VPS_HOST }}
    username: ${{ secrets.VPS_USER }}
    key: ${{ secrets.VPS_SSH_KEY }}
    source: "."
    target: "~/StockMood"
```

```
- name: Deploy via SSH
  uses: appleboy/ssh-action@master
  with:
    host: ${{ secrets.VPS_HOST }}
    username: ${{ secrets.VPS_USER }}
    key: ${{ secrets.VPS_SSH_KEY }}
    script: |
      cd ~/StockMood
      docker compose down
      docker compose up -d --build
```

Result: Push → GitHub Connects → Server Updates

Inside bash-

```
git push origin main
```

GitHub:

Uses your private SSH key to access the server

Copies the latest files to `~/StockMood`

Runs Docker commands to rebuild and restart your app

Visual Summary:

```
[Your PC] → git push →
    [GitHub Actions]
        ↓ (SSH using private key)
    [Your Server (Hostinger)]
```

```
        ↓
    docker compose up -d --build → 🚀 App goes live
```

so, when i need to update a change then-git-cd "C:\Users\madha\OneDrive\Desktop\StockMood"

```
git add frontend/WaterWhizFrontend/src/pages/Home.vue

git commit -m "Update Home.vue banner layout"

git push origin main
```

-

```
# After you change any file

git add .

git commit -m "My change"

git push origin main
```

path-

1. GoDaddy
2. Cloudflare
3. Hostlinger VPS
4. Fastpanel
5. Nginx conf
6. GitHub