### Deploying Flask Backend and Frontend on a Single Public URL Using Nginx

**1. Problem Statement**

The frontend is hosted on Hostinger and the backend (Flask API) is hosted on a VM.

A single public URL (https://yourdomain.com) is required for the entire application.

The backend should be accessible via /api/ instead of a separate subdomain.

Security is critical, and HTTPS (SSL) should be enabled.

The frontend and backend need CORS compatibility for API requests.

**2. Solution Overview**

The solution involves:

Using Nginx as a reverse proxy on the VM to route traffic efficiently.

Configuring Hostinger DNS to point yourdomain.com to the VM’s public IP.

Serving the frontend from Hostinger and forwarding backend API requests to Flask using Nginx.

Enabling HTTPS using Let’s Encrypt for secure communication.

Configuring Flask to handle API requests properly.

1. **Implementation Steps**

Step 1: Configure Hostinger DNS

Log in to Hostinger Dashboard → Domains → DNS Zone.

Add an A Record with:

**Host: @**

**Value: VM Public IP**

Now, yourdomain.com points to your VM.

Step 2: Install and Configure Flask on VM

Install Python and Flask

**sudo apt update && sudo apt install python3 python3-pip -y**

**pip3 install flask gunicorn flask-cors**

Create a Flask API (app.py)

**from flask import Flask, jsonifyfrom flask\_cors import CORS**

**app = Flask(\_\_name\_\_)**

**CORS(app, origins=["https://yourdomain.com"]) # Allow frontend access**

**@app.route('/api/data', methods=['GET'])def get\_data():**

**return jsonify({"message": "Hello from Flask API!"})**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(host='0.0.0.0', port=5000)**

**Run Flask with Gunicorn**

**gunicorn --bind 0.0.0.0:5000 app:app**

This ensures that the Flask API runs on port 5000.

Step 3: Install and Configure Nginx on VM

Install Nginx

**sudo apt install nginx -y**

**Create a New Nginx Configuration**

**sudo nano /etc/nginx/sites-available/yourdomain.com**

Add the Following Configuration:

nginx

**server {**

**listen 80;**

**server\_name yourdomain.com;**

**# Serve Frontend (React/Vue/Angular) from Hostinger**

**location / {**

**proxy\_pass https://frontend-hostinger.com;**

**proxy\_set\_header Host $host;**

**proxy\_set\_header X-Real-IP $remote\_addr;**

**proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;**

**}**

**# Serve Flask API on /api/**

**location /api/ {**

**proxy\_pass http://127.0.0.1:5000/;**

**proxy\_set\_header Host $host;**

**proxy\_set\_header X-Real-IP $remote\_addr;**

**proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;**

**}**

**}**

->This configuration does the following:  
✅ Requests to yourdomain.com serve the frontend from Hostinger.  
✅ Requests to yourdomain.com/api/ route to the Flask backend on the VM.

->Enable the Configuration and Restart Nginx

**sudo ln -s /etc/nginx/sites-available/yourdomain.com /etc/nginx/sites-enabled/**

**sudo systemctl restart nginx**

Now, Nginx properly routes frontend and backend requests.

Step 4: Secure API with HTTPS (SSL)

Install Certbot and Obtain SSL Certificate

**sudo apt install certbot python3-certbot-nginx -y**

**sudo certbot --nginx -d yourdomain.com**

Modify Nginx Configuration for HTTPS

Update /etc/nginx/sites-available/yourdomain.com to include SSL:

nginx

s**erver {**

**listen 80;**

**server\_name yourdomain.com;**

**return 301 https://$host$request\_uri;**

**}**

**server {**

**listen 443 ssl;**

**server\_name yourdomain.com;**

**ssl\_certificate /etc/letsencrypt/live/yourdomain.com/fullchain.pem;**

**ssl\_certificate\_key /etc/letsencrypt/live/yourdomain.com/privkey.pem;**

**location / {**

**proxy\_pass https://frontend-hostinger.com;**

**proxy\_set\_header Host $host;**

**proxy\_set\_header X-Real-IP $remote\_addr;**

**proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;**

**}**

**location /api/ {**

**proxy\_pass http://127.0.0.1:5000/;**

**proxy\_set\_header Host $host;**

**proxy\_set\_header X-Real-IP $remote\_addr;**

**proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;**

**}**

**}**

Restart Nginx

sudo systemctl restart nginx

Now, your site is accessible securely via HTTPS.

Step 5: Update Frontend to Fetch API Data

Modify the frontend to call the API on the same domain (yourdomain.com/api/):

javascript

**fetch("/api/data")**

**.then(response => response.json())**

**.then(data => console.log(data))**

**.catch(error => console.error("Error fetching data:", error));**

Step 6: Final Testing and Validation

Open [https://yourdomain.com](https://yourdomain.com" \t "_new) and verify the frontend loads.

Check if the frontend successfully fetches data from https://yourdomain.com/api/.

Use browser developer console (Network Tab) to confirm API requests.

4. Final Setup Recap

| ***Component*** | ***URL*** | ***Description*** |
| --- | --- | --- |
| ***Frontend*** | ***https://yourdomain.com*** | ***Hosted on Hostinger*** |
| ***Backend (Flask)*** | ***https://yourdomain.com/api/*** | ***Hosted on VM*** |
| ***Nginx Reverse Proxy*** | ***https://yourdomain.com*** | ***Routes requests properly*** |
| ***SSL Security (HTTPS)*** | ***https://yourdomain.com*** | ***Enabled via Certbot*** |

Research By :

Mansi Patel

Ganesh Kalyankar