

Deploying APIs with custom domains with Nginx, Let's Encrypt, FastAPI, and Uvicorn



Chirag Sharma · [Follow](#)

13 min read · Dec 17, 2023



20



1





Goals

- The end goal is to deploy locally hosted APIs to the AWS EC2 instance and access them via a domain. For example, you might be using

localhost:3000 to run your FastAPI app, this guide will help you run it on an EC2 instance and access it via your own purchased domain for example chiragsharma.xyz

Secondary Goals

- Get your hands with end to end flow of setting up all things web-related.
- This guide is not designed to be deep into individual components, it gets you aware of all the moving parts of the web to achieve our end goal. Once you have followed this guide to the end, you will find it easy to deep dive into one of each component as you will now be aware of where they fit.

Summary

This guide is broken down into the following sections:

- **Local Development:** This is just to get ourselves going, if you are handsy with Python you can directly start by *Setting up our code on the AWS EC2 Instance* section.
- **Setting up our code on AWS EC2 Instance:** This section deals with launching a new AWS EC2 Instance, Adding an Elastic IP to it, and then installing dependencies and running our code making it accessible via our Elastic IP or public IP DNS.
- **Purchasing a Domain and Mapping IP to DNS:** This section deals with purchasing a domain(Namecheap in my case) and then mapping our Elastic IP to DNS in Domain Name Registrar.
- **Configuring SSL Certificates and Nginx as Reverse Proxy:** This section deals with installing SSL certificates from Let's Encrypt and then setting

up our Nginx as a reverse proxy so that we can use our domain to access our endpoints.

Who is this for?

- I have tried to make this guide as simple as possible to follow but if you have prior experience with FastAPI and basic AWS EC2 Instances you will breeze through the guide, if not you have to start learning the concepts why not now?

Local Development

Quick Discussion on FastAPI and uvicorn

Make sure you have **python3.10** installed, we are choosing python3.10 as this is the default python version that comes configured in Ubuntu Instances on AWS right now. To keep it simple let's go with this version on our local too.

We will be building this project on top of FastAPI and uvicorn:

What is FastAPI?

- In short, it's a web framework with totally awesome features right out of the box, you have fast performance, automatic API documentation, solid data validation, and serializations inbuilt. It's very fast to test and spin-off web APIs on FastAPI.

Why are we using it?

- Quick, fast, large community support.

- Microsoft, Uber, and Netflix are using FastAPI in their internal or external work.

What's uvicorn?

- Uvicorn is a tool that helps your Python web application talk to the internet. Think of it as a middleman who takes requests from users on the internet, like when someone visits a web page and hands them over to your application. Then, it takes the response from your application (like the content of a web page) and delivers it back to the user's browser.

Why are we using it?

- It's fast, asynchronous, lightweight, and works well with modern Python web apps.

Setting up Code and Project

Let's start with the local development. We will be building dummy API Endpoints.

Once you have cloned the [GitHub Repo](#), create a python3.10 virtualenv and install the requirements.txt in it. Make sure the virtualenv is activated and you are in the project directory. You are ready to proceed to the next steps.

- Type ``uvicorn main:app -- reload`` in the terminal and let the system roll if everything goes well you will start seeing the following logs in the terminal with it being under the control of uvicorn.

```
INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:      Started reloader process [19320] using StatReload
INFO:      Started server process [30656]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
```

- If your system is correctly up until this point, you can run the following cURLs and expect the following response.

```
curl --location 'localhost:8000'

Response 200 OK:
{
  "message": "Endpoint is up!!"
}
```

```
curl --location 'localhost:8000/me'

Response 200 OK:
{
  "message": "This endpoint is about my handle @plusminuschirag go find me on
```

Code Explanation

Project Directory has only single Python file \Rightarrow main.py which has the following contents. The file code is pretty straightforward with most basic FastAPI usage.

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Endpoint is up!!"}

@app.get("/me")
async def me():
    return {"message": "This endpoint is about my handle @plusminuschirag go find me"}
```

Setting up our code on AWS EC2 Instance.

- The goal of this section is to create an AWS EC2 Instance and run this code on the server in a way that it doesn't stop if you close the SSH tunnel to your server.

Creating an EC2 Instance

- Login to your AWS Account, go for [EC2 Dashboard](#). Click on Instances(Running) Tile in the middle of your screen under Resources Block.
- Click on Launch instances in the top right corner, it's an orange button very easy to spot.
- You will be taken to a Launch an Instance Page where you will see some configurations as you scroll down. So scroll back to the top, and let's start filling them one by one.

Name

- Fill in any name for the server that is easily recognizable for you. For example, I used chiragsharma.xyz to highlight that this is the URL I will be hosting on the server, name should be easily clickable for you.

Application and OS Images

- Select Ubuntu from the “Under the Quick Start”. Let other options be as they are, you don’t need to change anything here.

Instance Type

- Select the t2.micro if it isn’t already selected, this comes under Free Tier if you have a new account or haven’t already exhausted the limit. You can use this free tier to follow this guide.

Key Pair(login)

- for SSHing into the server. If you don’t know what they are it pays to read about it. If you know create a new key pair, name it something conventional for you, download the keys, and keep them safe, if you lose them your server access is as good as gone.

Network Settings

- Tick the Checkbox for Allow HTTPS traffic from the internet
- Tick another checkbox for Allow HTTP traffic from the internet

Configure Storage

- You can leave it at 8 GiB, no need to change for this guide.

That's all the configuration you need to run your server, click `Launch instance` button on the right panel. It's an orange button again very easy to spot.

Once you click the button your instance will be created and then launched, go back to your Instances Dashboard and wait for the Instance State of your server to become `Running`

Adding Elastic IP to our EC2 Instance

Click on your `instance Id` from the dashboard and you will be taken to the instance summary page. Here you will see two panels:

- Top panel for instance summary
- Bottom panel with multiple tabs like Details, Security, Networking etc

Top Panel

- You will find `Public IPv4 DNS` and `Elastic IP` addresses
- **Public IPv4 DNS:** This is a DNS assigned to you by the AWS. This is dynamic, if you stop or restart your instance this can change, this will be very problematic we will see later on in our guide.
- **Elastic IP addresses:** You will find this field empty right now because AWS doesn't give you Elastic IP by default, you have to get an IP from the AWS console and assign it to your server. This will be static IP and you will hold it until you release it from your AWS. If you stop or restart your instance this IP will not change.

Go back to your EC2 Dashboard and click Elastic IPs from the middle top Resources Panel.

Click on the `Allocate Elastic IP address` button on the top right, let the settings by default, and click `Allocate` .

As soon as you do that you will have that IP, all that remains is to map it to our instance. You will see your newly generated Elastic IP in front of you click on the IP Address and you will be taken to another Summary page for this Elastic IP.

Click `Associate Elastic IP address` and select the name of your instance in the field, once you select that you will see private IP address filled with your private IP of the instance, if not then just click on it and select your private IP.

Tick the Checkbox for `Allow this Elastic IP address to be reassociated`

Now you have assigned a static IP to your EC2 Instance.

Running our code

Click the Instance ID of your server from your Instance Dashboard(left panel), Click the Connect Button in the Top Row.

Select the SSH Client Tab, here you will be given an example of how to connect to your server via SSH below

```
ssh -i "chiragsharma.xyz.pem" ubuntu@ec2-52-23-30-80.compute-1.amazonaws.com
```

This pem file is the same ssh key-value pair file you created while launching your instance. Navigate in your terminal to the directory where the pem file is present and directly run the command from the example on your SSH Client Tab.

If everything goes right you will be in your server.

- Your server right now is just a clean slate.

Let's do a step-by-step setup of the code on this server.

Run `sudo apt update`

- Before you install or upgrade packages, your system must know what the latest versions of packages are and where to download them from. This ensures you're getting the latest and potentially the most secure versions of the software.

Run `python3 --version`

- This should print Python 3.10.12 on your screen.

Run `git clone https://github.com/plusminuschirag/api-endpoint-domain-reverse-proxy-configuration.git`

- This will clone the repository in your server.

Run `ls -l` and it will show you list of directories present, you should see only one directory `api-endpoint-domain-reverse-proxy-configuration`

Run `mv -T api-endpoint-domain-reverse-proxy-configuration web-hosting`

- Let's change the folder name to `web-hosting`

Run `cd web-hosting` to move into the directory

Run `python3 -m venv venv` to create a virtualenv

- Ubuntu might give you some errors while doing this one but will show you the command directly on the screen so you can fix it up.

Run `source/bin/activate` to activate your virtualenv. If you run into the following error:

```
root@ip-172-31-26-10:/home/ubuntu/web-hosting/venv/bin# ./activate
bash: ./activate: Permission denied
```

Simply run `chmod +x source/bin/activate`

- Sometimes we don't have execution permission, so this command will add execution permission to the activated file.
- Once activated, run `pip install -r requirements.txt`
- This will install the Python pip packages in your environment.

Run `uvicorn main:app --host 0.0.0.0 --port 8000 --reload`

This will start the uvicorn server and if everything goes fine we will see the same logs as we saw in the local setup.

```
INFO:      Uvicorn running on <http://0.0.0.0:8000> (Press CTRL+C to quit)
INFO:      Started reloader process [19320] using StatReload
INFO:      Started server process [30656]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
```

We have done something differently here from the local setup, we are running on host 0.0.0.0 instead of 127.0.0.1. Passing host as 0.0.0.0 tells the uvicorn to allow requests from outside the local network.

You can now use your `<elastic-ip>:8000` to access the endpoints.

Wait what? It didn't work yes it will not. Check out the next section for that.

First Terminate the server by pressing CTRL + C.

Why it didn't work and making it work.

See it this way, you have hosted your app on port 8000 but your EC2 instance as of now doesn't allow anything to connect at port 8000.

We have to make your port, 8000 in this case to allow incoming traffic. For this, we have to add something called the Inbound rule to allow requests at 8000.

Follow the Steps and you will allow traffic on your **port 8000**:

- Go to your Instances Dashboard.
- Click on your Instance ID
- Select Security tab from the Bottom Panel
- Click on the security group from the Security Groups
- Click Edit Inbound Rule and a new rule table will open.
- Click on Add rule
- Select Custom TCP in Type
- Add 8000 to Port Range
- Right After Source Dropdown Select the empty field and click 0.0.0.0/0 from the dropdown.
- Click on the Save rules

You can now use your `<elastic-ip>:8000` to access the endpoints.

You can also make a curl request to the same just like you did on your local host.

Terminate the server by pressing CTRL + C.

Running our uvicorn server for practical purposes.

We cannot run our uvicorn server this way for the following reasons:

- This server will terminate as soon as we exit out of our SSH connection.

- This server blocks your terminal, you cannot do any other operation while it has occupied the terminal.

We can use several options to deal with this challenge:

nohup

- The **nohup** command can be used to run commands in the background and ignore the hang-up signal.

screen

- **screen** is a terminal multiplexer that enables you to manage multiple terminal sessions and keep processes running after you disconnect.

tmux

- Similar to **screen**, **tmux** allows you to detach from a session and reattach later.

systemd

- **systemd** is used for starting, stopping, and managing services and daemons. Services are defined in **.service** files, which **systemd** uses to control how and when to start these services.
- **systemd** is designed for parallel execution, which allows for faster boot times as it can start services simultaneously, rather than sequentially.
- **systemd** includes **journald**, a logging daemon that collects and manages system and service logs, providing advanced logging capabilities.

Setting up our systemd file

Run `sudo nano /etc/systemd/system/web-hosting.service`

This will create and open a file named “web-hosting.service” for you in the mentioned directory.

```
[Unit]
Description=Uvicorn instance to serve my app
After=network.target

[Service]
User=ubuntu
Group=ubuntu
WorkingDirectory=/home/ubuntu/web-hosting
ExecStart=/home/ubuntu/web-hosting/venv/bin/uvicorn main:app --host 0.0.0.0 --po
Restart=always
RestartSec=5
KillSignal=SIGQUIT
Type=simple
StandardError=syslog
NotifyAccess=all

[Install]
WantedBy=multi-user.target
```

Pay Attention to one thing, in our ExecStart entry we are mapping the uvicorn in the virtualenv, not activating virtualenv.

Double check you have the same content in the file, press `ctrl + s` and `ctrl + x` to save and exit respectively.

- Restart the systemctl daemon : `sudo systemctl daemon-reload`
- Restart the web-hosting service: `sudo systemctl start web-hosting.service`
- See the status of your service : `sudo systemctl status web-hosting.service`

By this point, your uvicorn server will keep running even if you close the SSH connection and your server will be running in the background

Purchasing a Domain

- If you are following this practical only, then buy a cheap domain, I bought mine from Namecheap for just \$3 a year, you can try other options. Purchasing a domain is very straightforward, for this section, I will be assuming you have bought the domain from Namecheap, if not? don't worry once you go through this section you will figure out what to do with yours.

Assuming you bought from NameCheap

- Once you have purchased your domain, go to your [Domain List](#)
- You will see your domain listed in front of you, select the drop-down on the right most end of the domain row and click Manage.
- Once it loads, click on the Advanced DNS Tab.
- Now you will see two default entries in your Host Records, you can delete them both.

We have to now create two new A Records entries:

- First Record

```
Type as A Record
Host @
Value : Elastic IP of your AWS Instance
TTL: 1 min
```

- Second Record

Type **as** A Record
Host **www**
Value Elastic IP of your AWS Instance
TTL : **1 min**

Not purchased from NameCheap

- You just have to locate your DNS records and add the above two entries in your DNS Records, you can delete the default records without problem.

Verifying your DNS Mapping

- After this step you have mapped your static IP with the domain name, so from now on whenever someone enters your domain, it will be resolved to the IP of the server you have just provided.
- You can go to [WhatsMyDNS](#) enter your domain, select A from the drop-down, and search wherever in the world your changes have been reflected, don't panic if they haven't it takes time to propagate through the world.

Configuring Certbot and Nginx

Till this point, you have completed the following:

- Hosted your app on EC2 instance which is accessible from outside.
- Linked your Domain Name to AWS Server IP in your Domain Name Registrar's DNS.

Our App is accessible using `<ELASTIC-IP>:<PORT>` right now but we don't see websites or APIs often working that way, they have domains instead. This section will deal with that.

We will allow our endpoint to handle http and https both traffic, for this, we need to have SSL certificates, for now, just understand https won't work without them. So now we have 2 things at our end:

- Get an SSL certificate
- Configure nginx to entertain http and https traffic.

SSH back into your server

Installing Certbot

- Run `sudo apt install certbot python3-certbot-nginx`
- Run `sudo certbot --nginx -d [yourdomain.com](<http://yourdomain.com/>) -d [www.yourdomain.com](<http://www.yourdomain.com/>)`
- In my case, `sudo certbot --nginx -d chiragsharma.xyz -d [www.](<http://www.yourdomain.com/>)chiragsharma.xyz`

Installing and Configuring Nginx

- Run `sudo apt install nginx`
- Run `sudo nano /etc/nginx/sites-available/yourdomain.com`
- In my case, `sudo nano /etc/nginx/sites-available/chiragsharma.xyz`
- Add the following to the file and save(`CTRL + S`) and exit(`CTRL + X`)

```

server {
    listen 443 ssl;
    server_name chiragsharma.xyz www.chiragsharma.xyz;

    ssl_certificate /etc/letsencrypt/live/chiragsharma.xyz/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/chiragsharma.xyz/privkey.pem;

    location / {
        proxy_pass http://localhost:8000;
        # Other necessary configurations...
    }
}

server {
    listen 80;
    server_name chiragsharma.xyz www.chiragsharma.xyz;
    return 301 https://$host$request_uri;
}

```

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

- A symbolic link named **yourdomain.com** is created in the **/etc/nginx/sites-enabled/** directory.
- This symlink points back to the original configuration file in **/etc/nginx/sites-available/yourdomain.com**.
- Nginx will now recognize and use the configuration for **yourdomain.com** when it starts or reloads, as it reads all configurations from the **sites-enabled** directory.
- In my case, I had to run `sudo ln -s /etc/nginx/sites-available/chiragsharma.xyz /etc/nginx/sites-enabled/`

Why Use Symbolic Links for Nginx Configuration:

- **Organization:** It keeps the actual configuration files in a single location (`sites-available`) while allowing you to easily enable or disable specific configurations by adding or removing symlinks in `sites-enabled` .
- **Flexibility:** This setup allows you to quickly activate or deactivate sites hosted on the server without having to move or edit the configuration files themselves.
- **Best Practice:** This method is a standard practice in managing Nginx configurations, providing clarity and control over which sites are currently active on the server.

Let's reload the nginx

- Run `sudo nginx -t`
- Run `sudo systemctl reload nginx`
- Your APIs are up, load your domain in the web browser.

Tada 🌟 and it's done.

- You can now put `[yourdomain.com](<http://yourdomain.com>)` in a web browser and will then see the relevant URL loading.
- In my case: `https://chiragsharma.xyz` became accessible.

Backend

Software Development

Python

AWS

API



Written by Chirag Sharma

71 Followers · 96 Following

Follow

2x Founding Engineer | Worked In Ed-Tech, Ad-Tech and FinTech

More from Chirag Sharma



Chirag Sharma

Genetic Algorithms—I

What if AI learns to bat with genes and their strategies?

Aug 9, 2019 🖱 315



Chirag Sharma

Influencer AI—A Giant in Making

Influencer AI is a term that got the hype around it in 2018, at the same time when the...

Oct 23, 2020 🖱 163



Open in app ↗

Sign up

Sign in

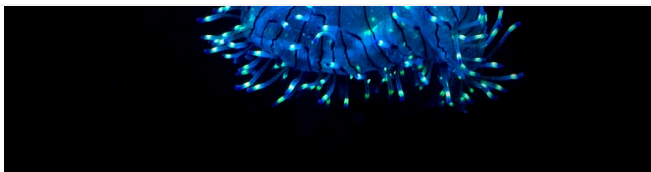
Medium



Search



Write



Chirag Sharma

Genetic Algorithms—2



In Towards Data Science by Chirag Sharma

Children or Parents who will do better, asked the wise men.

Aug 20, 2019 🖱 315



Simplest Guide for Regular Expressions: NLP Made Easy

People often avoid learning regular expressions and then they end up without...

May 20, 2020 🖱 215



See all from Chirag Sharma

Recommended from Medium



Harendra

How I Am Using a Lifetime 100% Free Server

Get a server with 24 GB RAM + 4 CPU + 200 GB Storage + Always Free

★ Oct 26 🖱 3.8K 💬 47



Fadi Shaar

Configuring NGINX Proxy for MinIO Server Using Docker-Compose

MinIO is a high-performance, S3-compatible object storage system. When deploying Minl...

★ Aug 8 🖱 11



Lists



Coding & Development

11 stories · 894 saves



Predictive Modeling w/ Python

20 stories · 1649 saves



General Coding Knowledge

20 stories · 1715 saves



Stories to Help You Grow as a Software Developer

19 stories · 1464 saves

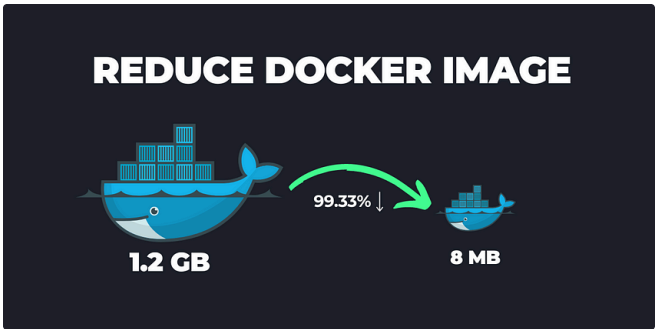



 In Stackademic by Abdur Rahman

Python is No More The King of Data Science

5 Reasons Why Python is Losing Its Crown

★ Oct 22 🖱️ 4.6K 💬 23 



 In AWS in Plain English by Dipanshu

Docker pros are shrinking images by 99%: The hidden techniques yo...

Unlock the secrets to lightning-fast deployments and slashed costs—before yo...

★ Sep 18 🖱️ 3.2K 💬 14 



☐ Benjamin Franklin. S

☐ Abhishek koserwal

Streamlit with Nginx: A Step-by-Step Guide to Setting up Your Dat...

Streamlit is an open-source app framework that is perfect for machine learning and data...

Jun 29  3



Securing FastAPI mTLS with Self-Signed Certificates

Securing APIs is crucial in today's digital world, especially when handling sensitive...

 Oct 1  3



See more recommendations