Deploy your FastAPI API to AWS EC2 using Nginx

Laura Calcagni · Follow
7 min read · Dec 12, 2021

FastAPI is an excellent tool for putting your machine learning models into production. In this article, I briefly explain how you can easily put your FastAPI in production to an AWS EC2 instance using Nginx.

FastAPI fundamentals

From the FastAPI website:

> *FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.*

Minimal code example

First things first, let's install FastAPI:

```
pip install fastapi
```

```
from fastapi import FastAPI

# Instantiate the class
app = FastAPI()

# Define a GET method on the specified endpoint
@app.get("/")
def hello():
    return {"result": "Welcome to FastAPI"}
```

main.py hosted with ❤ by GitHub view raw

To start, we import the fastAPI class from the fastapi module and then instantiate this class creating the object app. Then, we define a function that returns a simple message in JSON format. This function has a decorator that defines a GET method on the specified path.

Running your app

Before running your app, you also need to install uvicorn, which is a lightweight server implementation to run our API.

```
pip install uvicorn
```

Now we are ready to run the application "app" located in the main.py file using the following command:

```
uvicorn main:app
```

By default, the API will be available in *http://127.0.0.1:8000* .

Automatic robust documentation

One of the characteristic things about FastAPI is that relies heavily on type hints for its capabilities. It uses Pydantic (a Python library for data parsing and validation) and standard type hints to create and check data models that allow you to automatically create robust documentation of the API.

By default the documentation is located at your API domain/docs. If you are running it locally, it will be available in *http://127.0.0.1:8000/docs* (provided by Swagger UI).
Alternatively, another automatic documentation provided by Redoc will be available in *http://127.0.0.1:8000/redoc* .

Some words about Nginx

Nginx is an open-source Web Server written in C that was designed with the purpose of being the world's fastest Web Server. It was created in 2004 by Igor Sysoev.
A Web Server is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests.
In addition to Nginx, there are other well-known web servers such as Apache, or IIS.
Nginx can also be used as a reverse and forward Proxy, a load balancer as well as other things as an API Gateway.

Why use Nginx?

Regardless of using uvicorn, exposing our API with Nginx has several advantages, but the one that will be most highlighted in this article is the ability to easily add an SSL certificate.

Deployment steps

The deployment process includes the following steps:

1. Create and launch the AWS EC2 instance.
2. Configure the AWS EC2 instance by installing Nginx and the API requirements
3. Configure Nginx.
4. Add an SSL certificate using OpenSSL.

Create and launch the AWS EC2 instance.

After login to your AWS account go to:
Services -> Compute ->EC2 ->Lauch Instance

Now you have to follow these steps:

Step 1: Choose an Amazon Machine Image (AMI)
I chose an Ubuntu 18.04 Server (note that it is Free Tier eligible).

Step 2: Choose an Instance Type
I chose the following, which is also Free Tier eligible.

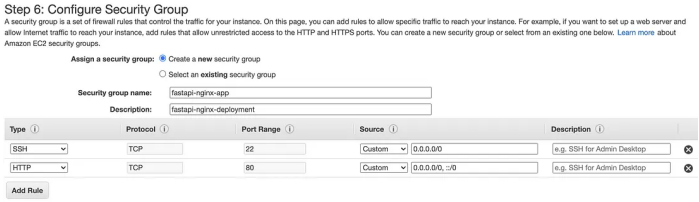
Currently selected: t2.micro (- ECUs, 1 vCPUs, 2.5 GHz, -, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GiB)	EBS-Optimized Available	Network Performance	IPv6 Support
	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
☑	t2	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes

Then, we leave the default settings for the following steps:

Step 3: Configure Instance Details
Step 4: Add Storage
Step 5: Add Tags

Step 6: Configure Security Group
By clicking on “Add Rule”, we will make sure to add the HTTP and HTTPS types



Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Learn more about Amazon EC2 security groups.

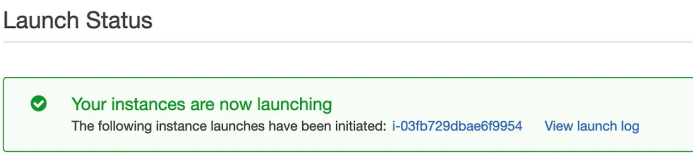
Assign a security group: Create a new security group
Select an existing security group

Security group name: fastapi-nginx-app
Description: fastapi-nginx-deployment

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
HTTP	TCP	80	Custom 0.0.0.0/0, ::/0	e.g. SSH for Admin Desktop

Add Rule

Step 7: Review Instance Launch
Finally, we review and launch the instance. When you click on the “Launch” button you will be prompted to create a key pair. Create and download it. You will need it to access the instance you have just created.

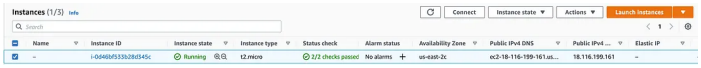


Launch Status

Your instances are now launching
The following instance launches have been initiated: i-03fb729dbae6f9954 View launch log

Configure the AWS EC2 instance by installing Nginx and the API requirements

Now that the instance is up and running, we will access it via SSH and configure it. To do it, go to Services -> Compute ->EC2 ->Instances
You should see your instance running:



Instances (1/3) Info

Connect | Instance state | Actions | Launch instances

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
-	i-0d46bf533b28d545c	Running	t2.micro	2/2 checks passed	No alarms	us-east-2c	ec2-18-116-199-161.us...	18.116.199.161	-

Select the instance and go to Actions -> Connect -> SSH Client

You will find there a detailed example of how to connect to your instance via SSH. In my particular case (as I named the key pair “fastapi-nginx.cer” and downloaded it to my “Downloads” folder) I will use this command to access my instance:

```
ssh -i Downloads/fastapi-nginx.cer ubuntu@ec2-18-116-199-161.us-east-2.compute.amazonaws.com
```

Congrats, you have just accessed to your EC2 instance via SSH:

```
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-1058-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sat Dec 11 23:06:08 UTC 2021

  System load:  0.02               Processes:           95
  Usage of /:   15.1% of 7.69GB    Users logged in:     0
  Memory usage: 20%                IP address for eth0: 172.31.45.46
  Swap usage:   0%


0 updates can be applied immediately.

New release '20.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Sat Dec 11 23:05:01 2021 from 45.176.88.102
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-45-46:~$
```

Now let's clone the repository I have prepared for this tutorial. You will see that it contains a minimal example of an API made with fastAPI.

```
git clone https://github.com/lcalcagni/Deploying-FastAPI-using-Nginx.git
```

Enter to the directory of the project to install the requirements:

```
cd Deploying-FastAPI-using-Nginx
sudo apt-get update
sudo apt install python3-pip
pip3 install -r requirements.txt
```

Let's run the API locally to check that everything is ok:

```
python3 -m uvicorn main:app
```

You will get something like this:

```
ubuntu@ip-172-31-45-46:~/Deploying-FastAPI-using-Nginx$ python3 -m uvicorn main:app
INFO:     Started server process [7675]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

So now let's make this API accessible for the rest of the world using Nginx.

Nginx configuration

First, install Nginx using the following command:

```
sudo apt install nginx
```

We already have the fastAPI API we wish to serve, now we need to create the server blocks that will tell Nginx how to do this.

By default, Nginx contains one server block called `default`. You can find it in this location: **etc/nginx/sites-enabled**

```
ubuntu@ip-172-31-45-46:/$ cd etc/nginx/sites-enabled
ubuntu@ip-172-31-45-46:/etc/nginx/sites-enabled$ ls
default
```

But we will create a new one called “fastapi_nginx” (you can choose another name):

```
cd /etc/nginx/sites-enabled/
sudo nano fastapi_nginx
```

Inside this file, we have to specify the following:

```
server {
    listen 80;
    server_name 18.116.199.161;
    location / {
        proxy_pass http://127.0.0.1:8000;
    }
}
```


18.116.199.161
With this configuration, you are telling Nginx, that what you have running on http://127.0.0.1:8000 should be served to http://18.116.199.161/ (port 80).

We save that file (Ctrl X) and then run

```
sudo service nginx restart
```

Then, we run the API

```
cd path/to/Deploying-FastAPI-using-Nginx
python3 -m uvicorn main:app
```

Then, try to access **http://{your EC2 public IP}**/using your browser on your local computer. You should see something like this:

Congrats! now your API is accessible for the rest of the world.

Add a self-signed SSL certificate using OpenSSL

Install OpenSSL and create the /etc/nginx/ssl directory:

```
sudo apt-get install openssl
cd /etc/nginx
sudo mkdir ssl
```

Then, we create the self-signed SSL certificate using this command:

```
sudo openssl req -batch -x509 -nodes -days 365 \
-newkey rsa:2048 \
-keyout /etc/nginx/ssl/server.key \
-out /etc/nginx/ssl/server.crt
```

After that, we add this certificate to our server block configuration:

```
cd /etc/nginx/sites-enabled/
sudo nano fastapi_nginx
```

Inside the file we make the following modification:

```
server {
    listen 80;
    listen 443 ssl;
    ssl on;
    ssl_certificate /etc/nginx/ssl/server.crt;
    ssl_certificate_key /etc/nginx/ssl/server.key;
    server_name 18.116.199.161;
    location / {
        proxy_pass http://127.0.0.1:8000;
    }
}
```

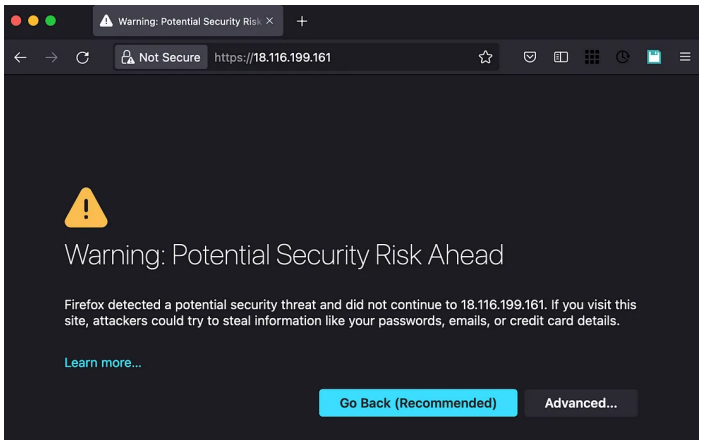
We save that file (Ctrl X) and then restart Nginx:

```
sudo service nginx restart
```

Finally, we run our API:

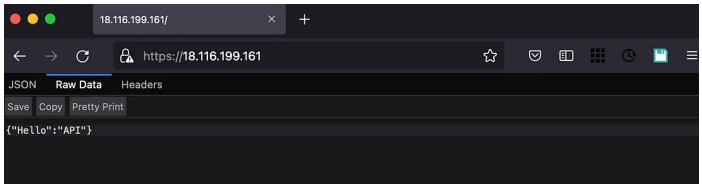
```
cd path/to/Deploying-FastAPI-using-Nginx
python3 -m uvicorn main:app
```

If everything works correctly, you should now be able to access your server over HTTPS (**https://{your EC2 public IP}/**). Your web browser (in this case I am using Firefox) may display a warning like this:



This is expected because of the type of certificate (self-signed) you are using. So you will have to manually confirm that you trust the server in order to access it.

Once you confirm that by clicking on the Advanced button, you will see your API available on **https://{your EC2 public IP}/**:



Notice that it is possible to redirect the HTTP to HTTPS adding this to the server block configuration (for more information check this):

```
return 301 https://$server_name$request_uri;
```

Don't forget to restart Nginx to apply the changes:

```
sudo service nginx restart
```

I hope you find this article useful. If you have any queries you can find me on LinkedIn.

Happy Deploying!

Laura Calcagni
Software Data Engineer

Fastapi | Nginx | Data Science | Ec2 | AWS

Written by Laura Calcagni

Follow

63 Followers

Software Data Engineer

More from Laura Calcagni

Laura Calcagni

Running R scripts in Airflow using Airflow BashOperators

In this article, I briefly explain how you can easily create a pipeline that executes a set o...

8 min read · Apr 6, 2021

203 3

Laura Calcagni

How to manipulate Google spreadsheets using Python

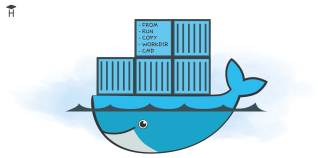
In this article, I briefly explain how you can easily manipulate your Google spreadsheet...

4 min read · Dec 31, 2021

22 1

See all from Laura Calcagni

Recommended from Medium



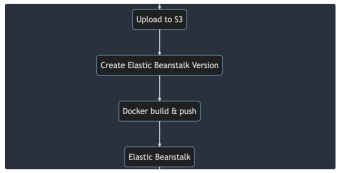
Utkarsha Bakshi in Geek Culture

Creating an AWS Lambda Python Docker Image from Scratch

In my previous tutorial, we learned how to use public.ecr.aws/lambda/python:3.8 as a base...

3 min read · Jan 5

23



Mai Văn Khánh in AWS Tip

Deploying a REST API Nodejs (NestJS) app with Github Actions...

Hi everyone! I started from scratch and made a new version with respect to deploying a...

12 min read · Mar 25

19 1

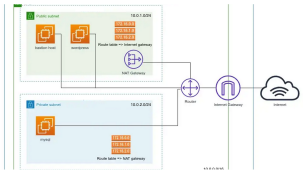
Lists

Predictive Modeling w/ Python
18 stories · 7 saves

New_Reading_List
173 stories · 1 save

Coding & Development
11 stories · 2 saves

Practical Guides to Machine Learning
10 stories · 21 saves



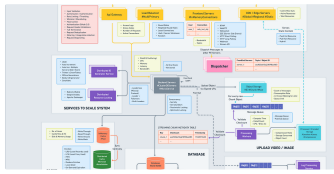
headintheclouds in AWS Tip

Deploy WordPress and MySQL with Terraform on AWS EC2 Like a Pro

Prerequisites:

14 min read · Jan 3

21 2



Love Sharma in Dev Genius

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However,...

9 min read · Apr 20

3.8K 23

Utkarsha Bakshi in Geek Culture

How to Dockerize a Python AWS Lambda Function

In this post, we will learn how to dockerize a Python AWS Lambda function and deploy it...

3 min read · Jan 5

68 4

Utkarsha Bakshi in Geek Culture

How to use OpenCV on AWS Python Lambda?

Learn how to run OpenCV on an AWS Python Lambda function.

5 min read · Mar 15

15

See more recommendations