Hosting Django project on AWS EC2 machine while Dockerizing and Config Nginx Server and SSL

Prepared by Ashish Kushwaha

Outline –

Create EC2 Instance -> Clone Repo -> Dockerize -> Get Domain address -> configure Nginx -> Add SSL

1. **Launch an EC2 Instance and connect to it.**
2. **Install the git**
    a. sudo apt-get update
    b. sudo apt-get install git
3. **Clone your repo**
    a. git clone https://github.com/your-username/your-repo.git
    b. cd your-repo
4. **Install Docker**
    a. Update the package list
        i. sudo apt-get update
    b. Install Docker
        i. sudo apt-get install docker.io
    c. Start Docker
        i. sudo systemctl start docker
    d. Configure Docker to start on boot
        i. sudo systemctl enable docker
5. **Create a Docker file**
    a. Navigate to the directory where you cloned your Django project code:
        i. cd /path/to/your/Django-project
    b. Create a file named ‘Dockerfile’:
        i. sudo nano Dockerfile

c. Copy paste the code attached with this Documentation in the Dockerfile. Replace your project-name in the Docker file.
d. Save and Close the 'Dockerfile' with Ctrl+O (save) and Ctrl+X (exit) [for nano Editor].

6. **Build the Docker image for your Django application.**
    a. Navigate to the directory where your 'Dockerfile' is located:
        i. cd /path/to/your/Django-project
    b. Build the docker image:
        i. sudo docker build -t <your-image-name> .
        (replace the <your-image-name>, the " . " (dot) at the end specifies that the build context in current directory.)
7. **Start the Docker container from the image we created.**
    a. Replace your-image-name with the name of your Docker image you created in the previous step.
        i. sudo docker run -p 8080:8000 -d <your-image-name>
    b. Check it the container is up and running by the command–
        i. Sudo docker ps

Make sure that your Docker container and Nginx server is running at different ports number. Other wise you will get port error for nginx.

Port 80 is default for Nginx server, that's why we have mapped port 8080 for docker conatainer.

This command starts a Docker container and maps port 8080 on the host to port 8000 inside the container. The **-d** flag runs the container in the background (detached mode).

You can check that the container is running using the **docker ps** command. The output should show your container along with its container ID, image, command, and status.

If the container is running correctly, you should be able to access your Django application by navigating to **http://<your-ec2-instance-public-ip>:8080** in your web browser.

If you are getting error –

unable to start container process: exec: "gunicorn":

add ‘gunicorn’ to your ‘requirements.txt’ file,

>> gunicorn==<version>

Replace <version> with the version of gunicorn that you want to install. You can find the latest version of gunicorn on the Python Package Index (PyPI) website.

Once you've added gunicorn to your requirements.txt file, you'll need to rebuild your Docker image using the docker build command to ensure that gunicorn is installed in the container. (Step 6 and 7).

This site can’t be reached 18.###.###.##8 (your EC2 public ip) took too long to respond.

This error message suggests that the web browser was not able to establish a connection to your EC2 instance at IP address 18.###.###.##8.

When you launch an EC2 instance, it is associated with a security group that acts as a virtual firewall to control the inbound and outbound traffic to the instance. By default, the security group allows all outbound traffic, but blocks all inbound traffic.

In order to allow inbound traffic to your instance, you'll need to configure the inbound rules of the security group. Here's how you can do it:

1. Go to the EC2 console and select the instance that you want to configure.
2. Under the "Description" tab, you'll see a section called "Security groups". Click on the security group associated with your instance.
3. In the "Inbound rules" tab, click on the "Edit inbound rules" button.
4. Click on the "Add rule" button to add a new inbound rule.
5. Select the type of rule you want to add. For example, if you want to allow HTTP traffic, select "HTTP" from the "Type" dropdown menu.
6. Specify the source of the traffic. You can either allow traffic from a specific IP address or range of IP addresses, or you can allow traffic from anywhere by selecting "Anywhere" from the "Source" dropdown menu.
7. Click on the "Save rules" button to save the new inbound rule.

After you've added the inbound rule, your EC2 instance should be able to receive traffic on the specified port.

Invalid HTTP_HOST header: '18.###.###.##8'. You may need to add '18.###.###.##8' to ALLOWED_HOSTS.

This error message indicates that the value of the HTTP_HOST header in the incoming HTTP request does not match any of the values in the ALLOWED_HOSTS setting of your Django application.

To fix this error, you need to add the IP address of your EC2 instance to the ALLOWED_HOSTS setting in your Django settings file. Here's how you can do it:

1. Open your Django settings file (settings.py) in a text editor.
2. Find the ALLOWED_HOSTS setting. It should be a list of strings.
3. Add the IP address of your EC2 instance to the list. For example, if the IP address is 18.###.###.##8, you should add '18.###.###.##8' to the list.
4. Save the file and restart your Django application.

After making this change, your Django application should be able to handle requests from the specified IP address.

Stop the running docker container

1. List the running container
    a. sudo docker ps
2. Stop the container
    a. sudo docker stop <container ID or name>
3. remove docker container (optional)
    a. sudo docker rm <container_name>

Again, rebuild the docker image and start the container from it. (Step 6 and 7).

Now, your website will be up and running at your EC2 ip address with (http://<ip-address>)

8. **Get a Domain name for your EC2 instance ip address. And now will have a accesss to your Django project at your Domain.**
9. **Configure Nginx server.**
    a. Install Nginx on your EC2 instance by running the following command:
        i. sudo apt-get update
        ii. sudo apt-get install nginx
    b. Start Nginx using the following command:
        i. sudo systemctl start nginx

c. Enable Nginx to start at boot time using the following command:
    i. sudo systemctl enable nginx
d. Edit the Nginx configuration file using the following command:
    i. sudo nano /etc/nginx/nginx.conf

e. In the http block, add the following lines:

```
server {
  listen 80;
  server_name <your-domain-name>;  (mywebsite.com)
  location / {
    proxy_pass http://localhost:8080;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
  }
}
```

Here, we are creating a virtual server block for our domain **<your-domain-name>** and proxying incoming traffic to our Docker container running on localhost:8000.

f. Save and close the file.
g. Test the Nginx configuration using the following command:
    i. sudo nginx -t
h. If the configuration is valid, reload Nginx using the following command:
    i. sudo systemctl reload nginx
i. With this configuration, Nginx should now be listening on port 80 and proxying incoming traffic to your Django application running on Docker.

10. Install SSL cert.

a. Obtain an SSL certificate from a trusted certificate authority (CA) like Let's Encrypt or purchase one from a commercial CA.
b. Install Certbot on your server. Certbot is a tool for automating the process of getting SSL/TLS certificates.
    i. sudo apt install certbot python3-certbot-nginx
c. Run the Certbot command to obtain an SSL/TLS certificate for your domain. The command will verify your domain and install the certificate on your server.
    i. sudo certbot --nginx -d example.com -d www.example.com

d. Certbot will ask you some questions like your email address and whether to redirect HTTP traffic to HTTPS. Answer the questions and let Certbot do its job.
e. Once Certbot has finished installing the SSL/TLS certificate, you can test your website by visiting https://example.com in your web browser.

11. **Make migrations in the server –**
    a. Go where your Dockerfile is present in the server and run the following command–
        i. Sudo docker exec -it <container-id> python manage.py makemigrations
        ii. sudo docker exec -it <container-id> python manage.py migrate
12. **Create a superuser in the server –**
    a. Go where your Dockerfile is present in the server and run the following command–
        i. Sudo docker exec -it <container-id> bash
    b. This will open a bash terminal, run the following commands (Enter the details, in the prompt.)
        i. Python manage.py createsuperuser
    c. Type "exit" to exit the bash terminal.
13. **Automate a container start (on rebooting the server) –**
    a. Sudo docker update -restart=always <container-id>