## Api installatiehandleiding

Deze handleiding is gebaseerd op deze tutorial:  
er zijn enkele kleine aanpassingen in dit document, die speciaal zijn voor onze flask applicatie.

<https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-16-04>

Het is het beste om deze handleiding uit te voeren om een ander account dan root, die wel root rechten heeft. Dit voorkomt fouten m.b.t rechten.

Als Apache2 op de server staat moet je dat eerst verwijderen, anders werkt nginx niet:

|  |
| --- |
| sudo apt-get purge apache2 |

Installeer python3, pip, libmysqlclient en nginx:

|  |
| --- |
| sudo apt-get update  sudo apt-get install python3  sudo apt-get install python3-pip python3-dev nginx  sudo apt-get install libmysqlclient-dev |

Als pip geïnstalleerd is kun je virtualenv installeren om een enviroment voor de applicatie te maken:

|  |
| --- |
| sudo pip3 install virtualenv |

Als dit alles gedaan is, kan de folder met de applicatie op de server gezet worden.

navigeer daarna naar de folder:

|  |
| --- |
| cd waterscan-api |

We gaan nu een environment maken voor de applicatie hier worden alle benodigde libraries geïnstalleerd:

|  |
| --- |
| sudo virtualenv venv |

We moeten nu de nieuw aangemaakte environment activeren dit doen we via de volgende commando:

|  |
| --- |
| source venv/bin/activate |

Nu de virtual environment geactiveerd is, en dat kunnen we zien omdat de terminal er nu anders uit ziet namelijk:

|  |
| --- |
| (venv) $ |

Is het tijd om de benodigde libraries te installeren. Dit gaat gemakkelijk omdat ze zijn opgenomen in de requirements.txt

|  |
| --- |
| sudo venv/bin/pip3 install -r requirements.txt |

Als alles goed geïnstalleerd is kunnen we de juiste poort open zetten, en de applicatie starten.

|  |
| --- |
| sudo ufw allow 8080 |

Als de poort open staat kunnen we de applicatie opstarten

|  |
| --- |
| gunicorn --bind 0.0.0.0:8080 wsgi:app |

Als de applicatie gewoon netjes opstart kunnen we door naar de volgende stap. We gaan een service aanmaken zodat Gunicorn automatisch opstart wanneer de server wordt opgestart.

We drukken op CTRL + C om de api te sluiten, en daarna typen we “deactivate” om de venv te sluiten.

|  |
| --- |
| deactivate |

## Create a systemd Unit File

The next piece we need to take care of is the systemd service unit file. Creating a systemd unit file will allow Ubuntu's init system to automatically start Gunicorn and serve our Flask application whenever the server boots.

Create a unit file ending in .service within the /etc/systemd/system directory to begin:

|  |
| --- |
| sudo nano /etc/systemd/system/waterscan-api.service |

Inside, we'll start with the [Unit] section, which is used to specify metadata and dependencies. We'll put a description of our service here and tell the init system to only start this after the networking target has been reached:

|  |
| --- |
| [Unit] Description=Gunicorn instance to serve waterscan-api After=network.target |

Next, we'll open up the [Service] section. We'll specify the user and group that we want the process to run under. We will give our regular user account ownership of the process since it owns all of the relevant files. We'll give group ownership to the www-data group so that Nginx can communicate easily with the Gunicorn processes.

We'll then map out the working directory and set the PATH environmental variable so that the init system knows where our the executables for the process are located (within our virtual environment). We'll then specify the commanded to start the service. Systemd requires that we give the full path to the Gunicorn executable, which is installed within our virtual environment.

We will tell it to start 3 worker processes (adjust this as necessary). We will also tell it to create and bind to a Unix socket file within our project directory called waterscan-api.sock. We'll set a umask value of 007 so that the socket file is created giving access to the owner and group, while restricting other access. Finally, we need to pass in the WSGI entry point file name and the Python callable within:

|  |
| --- |
| [Unit] Description=Gunicorn instance to serve waterscan-api After=network.target  [Service] User=ubuntu Group=www-data WorkingDirectory=/home/ubuntu/waterscan-api Environment="PATH=/home/ubuntu/waterscan-api/venv/bin" ExecStart=/home/ubuntu/waterscan-api/venv/bin/gunicorn --workers 3 --bind unix:waterscan-api.sock -m 007 wsgi:app |

Finally, we'll add an [Install] section. This will tell systemd what to link this service to if we enable it to start at boot. We want this service to start when the regular multi-user system is up and running:

|  |
| --- |
| [Unit] Description=Gunicorn instance to serve waterscan-api After=network.target  [Service] User=ubuntu Group=www-data WorkingDirectory=/home/ubuntu/waterscan-api Environment="PATH=/home/ubuntu/waterscan-api/venv/bin" ExecStart=/home/ubuntu/waterscan-api/venv/bin/gunicorn --workers 3 --bind unix:waterscan-api.sock -m 007 wsgi:app  [Install] WantedBy=multi-user.target |

With that, our systemd service file is complete. Save and close it now.

We can now start the Gunicorn service we created and enable it so that it starts at boot:

|  |
| --- |
| sudo systemctl start waterscan-api  sudo systemctl enable waterscan-api |

## Configuring Nginx to Proxy Requests

Our Gunicorn application server should now be up and running, waiting for requests on the socket file in the project directory. We need to configure Nginx to pass web requests to that socket by making some small additions to its configuration file.

Begin by creating a new server block configuration file in Nginx's sites-available directory.

|  |
| --- |
| sudo nano /etc/nginx/sites-available/waterscan |

Open up a server block and tell Nginx to listen on the port 5000. We also need to tell it to use this block for requests for our server's domain name or IP address.

The only other thing that we need to add is a location block that matches every request. Within this block, we'll include the proxy\_params file that specifies some general proxying parameters that need to be set. We'll then pass the requests to the socket we defined using the proxy\_pass directive:

|  |
| --- |
| server {  listen 8080;  server\_name 145.136.241.162;   location / {  include proxy\_params;  proxy\_pass http://unix:/home/ubuntu/waterscan-api/waterscan-api.sock;  } } |

That's actually all we need to serve our application. Save and close the file when you're finished.

To enable the Nginx server block configuration we've just created, link the file to the sites-enabled directory:

|  |
| --- |
| sudo ln -s /etc/nginx/sites-available/waterscan /etc/nginx/sites-enabled |

With the file in that directory, we can test for syntax errors by typing:

|  |
| --- |
| sudo nginx -t |

If this returns without indicating any issues, we can restart the Nginx process to read the our new config:

|  |
| --- |
| sudo systemctl restart nginx |

We moeten nu hetzelfde doen voor angular:

|  |
| --- |
| sudo nano /etc/nginx/sites-available/angular |

Kopieer de volgende code block naar het bestand:

|  |
| --- |
| server {  listen 80;  server\_name 145.136.241.162;  root /usr/share/nginx/html;  location / {  # First attempt to serve request as file, then  # as directory, then redirect to index(angular) if no file found.  try\_files $uri $uri/ /index.html;  }  } |

To enable the Nginx server block configuration we've just created, link the file to the sites-enabled directory:

|  |
| --- |
| sudo ln -s /etc/nginx/sites-available/angular /etc/nginx/sites-enabled |

With the file in that directory, we can test for syntax errors by typing:

|  |
| --- |
| sudo nginx -t |

If this returns without indicating any issues, we can restart the Nginx process to read the our new config:

|  |
| --- |
| sudo systemctl restart nginx |

The last thing we need to do is adjust our firewall again. We can then allow access to the Nginx server

|  |
| --- |
| sudo ufw allow 'Nginx Full' |

Dit was alles om de webserver op te zetten.  
Als dat nog niet gedaan is kun je nu angular installeren.