AAD / Backend Development

Host a dockerized application using HTTPS

Last changed: September 2023
Author: Jan Willem Boer

Contents

Create and publish docker image

After you build your project (sometimes also using a docker image), you can deploy the build output as docker image. For this tutorial, we'll use a very simple nodejs application, which will listen to port 3000 and show the word "hi" for a GET request at the root.

Prepare

Create an account at https://hub.docker.com first and choose the free tier. Confirm your email address before proceeding.

Also make sure you have docker installed.

Clone the repository at https://github.com/janwillemboer/dockerized-nodejs-app-for-aws

Build docker image

In the repository root, build the docker image:

```
> docker build -t your-docker-username/aad-demo .
```

Test the image:

```
> docker run --rm -d -p 3000:3000 --name aad-demo
    your-docker-username/aad-demo
```

There should be a cheerful "hi" at localhost:3000 when you try it in a browser.

Push the image to your docker repository

In the repository root, login with the credentials you just provided to docker:

```
> docker login
```

After this, push the image to your personal repository:

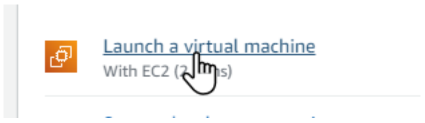
```
> docker push your-docker-username/aad-demo
```

Deploy docker image to AWS instance

Docker images can be deployed to AWS in several different ways. AWS has native support for hosting docker images in the Elastic Container Service. In this tutorial however, we will deploy the image on a normal AWS EC2 instance (a VM), because we don't want this to be too AWS-specific. But feel free to experiment with ECS as well.

Launch AWS EC2 instance

In AWS, scroll down to "Launch a virtual machine" and click it



- Give the VM a name, for example AAD Demo
- Select Ubuntu
- Keep t2.micro

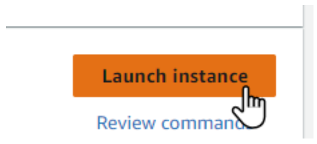
Key pair:

- Generate a key pair -> download and store the private key (the pem file) somewhere!
- Select the newly generated key

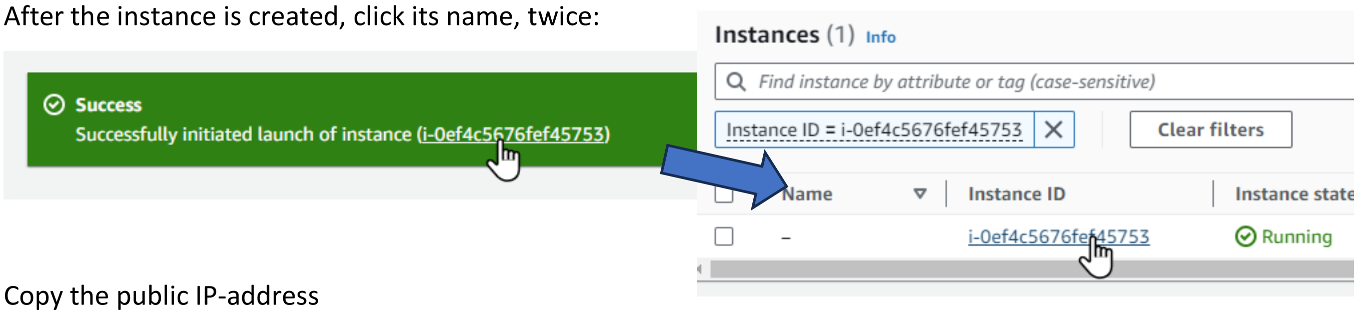
Network:

- Allow ssh traffic from all locations
- Allow https and http

Press "Launch instance" on the right.

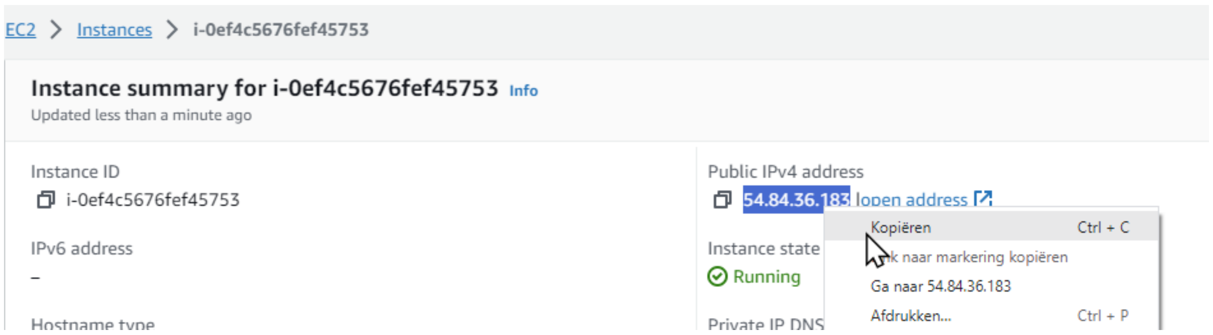


After the instance is created, click its name, twice:



Success
Successfully initiated launch of instance (i-0ef4c5676fef45753)

Copy the public IP-address



EC2 > Instances > i-0ef4c5676fef45753

Instance summary for i-0ef4c5676fef45753 Info
Updated less than a minute ago

Instance ID
i-0ef4c5676fef45753

IPv6 address
–

Hostname type

Public IPv4 address
54.84.36.183 | open address

Instance state
Running

Private IP DNS

Connect to instance

(Install openssh if you don't have it on your machine)

In your home directory, go to the .ssh directory.

- On windows: %userprofile%\.ssh
- On unixy-systems: ~/.ssh

Copy the .pem keyfile into this directory.

Open a terminal and SSH into your AWS virtual using the IP-address you just copied, using "ubuntu" as username and the keyfile:

```
> ssh -i "%userprofile%\.ssh\your-key-file.pem" ubuntu@54.84.36.183
```

(add the host to the trusted hosts by typing "yes")

Note: the IP-address will change each time the instance goes down and is restarted. See below on how to get a fixed IP-address.

Install docker

Once you are logged into the AWS box, install docker with

```
> sudo snap install docker
> docker --version
Docker version 20.10.24, build 297e128
```

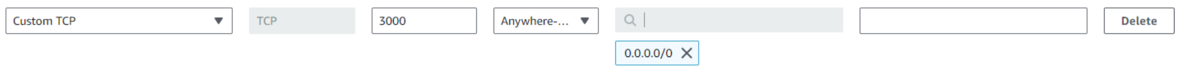
Start your docker image

Now start the image you just uploaded to the docker registry:

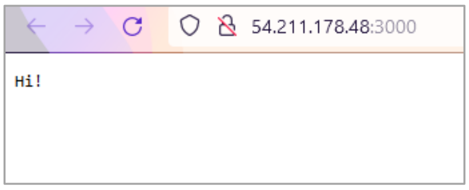
```
> sudo docker run -d -p 3000:3000 --name aad-demo
    your-docker-username/aad-demo
```

NOTE: the docker container will go down if the VM goes down, you will have to restart it manually each time. To prevent this, add it as a systemctl service (see Appendix: create a service for a docker container).

You can't test the server, because the virtual has firewall rules which don't allow port 3000 to be visited. If you want to test it, add a firewall rule to allow TCP traffic on port 3000 on the Security tab of the instance overview:


Custom TCP | TCP | 3000 | Anywhere-... | 0.0.0.0/0

After this, you should be able to visit the server on the public IP-address on port 3000:


54.211.178.48:3000

Hi!

Now delete the rule again, you don't want this port to be exposed.

Add reverse proxy

To reach the webapi on port 80, you'll need to add a reverse proxy, which forwards the traffic for port 80 to the port the webapi is listening on. Why not just let the api listen on port 80? Several reasons:

- Scalability. A proxy is also able to do load balancing and forward the traffic to another instance of the same api.
- HTTPS. A proxy can translate https to http traffic.
- Multiple apps on one server. A proxy can do routing to the right app based on the url (or domain name).
- Security. Having a proxy in front of the application may reduce attack vectors

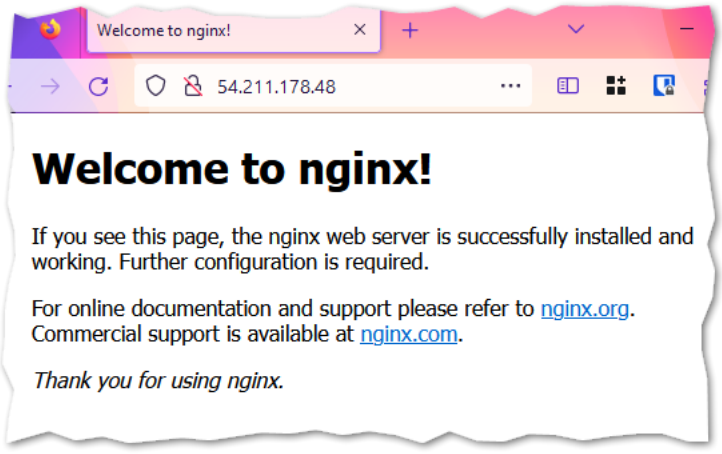
There are several options for reverse proxying: for example HAProxy and nginx (pronounced "engine X"). We'll use nginx in this example, because later on it works very easy in case you're adding TLS certificates for HTTPS.

Prepare

Install nginx on the AWS machine:

```
> sudo apt update
> sudo apt install nginx-core -y
```

You should now be able to load the default nginx webpage on the public IP of the AWS machine:



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

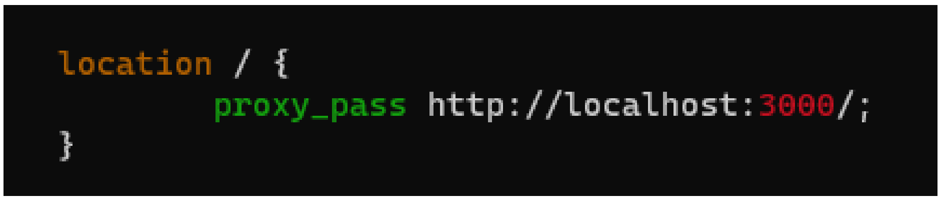
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Configure nginx to send traffic to the API

On the VM, sudo edit the file /etc/nginx/sites-available/default with your favorite editor (nano or vi).

Find the line that starts with "location /" and change it to this:

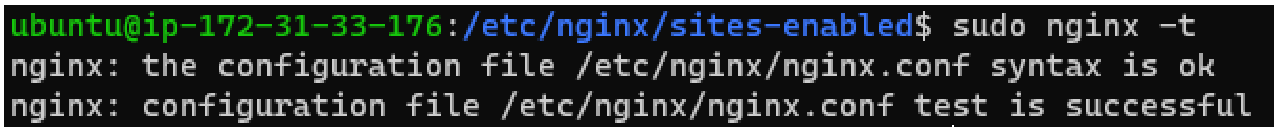


```
location / {
        proxy_pass http://localhost:3000/;
}
```

This will tell nginx to pass all traffic from port 80 to the local application listening on port 3000, which happens to be our own docker image.

Test the nginx config:

```
> sudo nginx -t
```

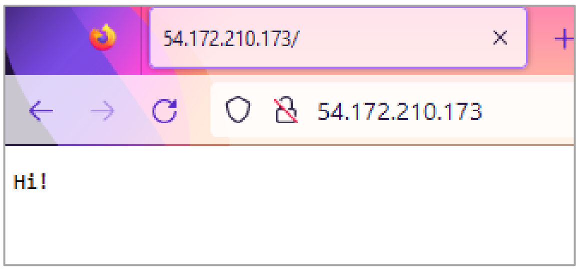


```
ubuntu@ip-172-31-33-176:/etc/nginx/sites-enabled$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

And restart nginx:

```
> sudo systemctl restart nginx
```

Now test the url to the server again. Instead of the default nginx page, we should now see our own “hi”-service:



54.172.210.173/

54.172.210.173

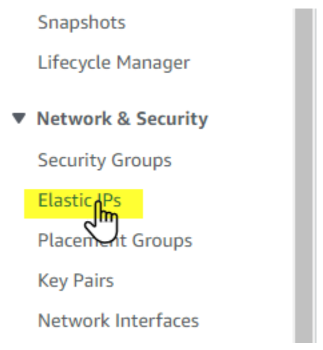
```
Hi!
```

Add domain name

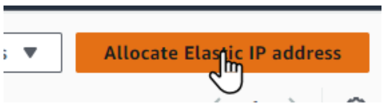
Instead of the ugly IP-address, it would be nice to have a domain name. This is only possible if the IP-address is fixed, which it isn't. But AWS has a possibility to get a fixed IP-address. This costs money, even when the lab is not running, but only a few cents per day.

Create fixed IP-address

In AWS, on the left, find "Elastic IPs" in the "Network and Security" tab:

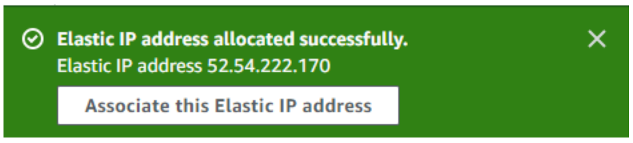


Click "Allocate Elastic IP address"



Just go with the defaults and click "Allocate".

It now says: "allocated successfully". Click "Associate this Elastic IP address" in this green bar:



Elastic IP address allocated successfully.
Elastic IP address 52.54.222.170

Select your instance and the private IP-address and click "Associate".

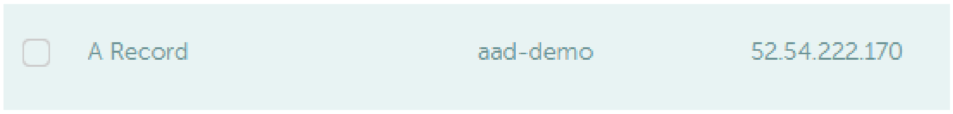
You can now use this IP-address to go to the server with your browser, and it should show the hi-message.

Note: don't forget that you also have to SSH to this address from now on.

Reserve domain name and add DNS records

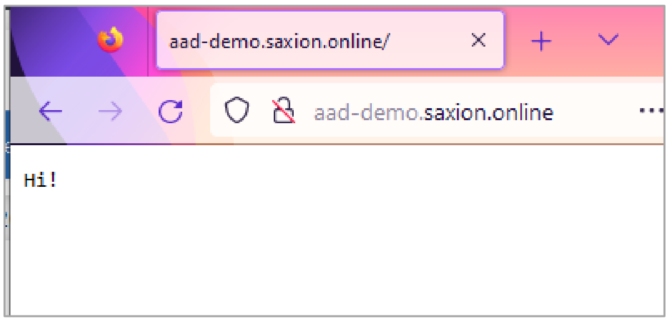
It is really nice to have your own domainname, and the simple ones only cost a few euros per year, so try it out if you can spare the euros! If not, ask your teacher for a subdomain on saxion.online: send them the Elastic IP-address and the required subdomain name (my-super-api.saxion.online, for instance).

If you do it yourself, add an A-record from your domain to the IP address. In this case the domain becomes aad-demo.saxion.online.


A Record aad-demo 52.54.222.170

Note: DNS changes can go very fast, but also can take a few hours to propagate. Check https://dnschecker.org/#A/ to see if your domain has been propagated to a DNS-server near you.

After the DNS changes have been done and propagated, you should be able to use the domain name to load the hi-service:



```
Hi!
```

Note: you can also use the domain name to SSH into your instance from now on.

Change nginx config to listen to this domain name specifically

Sudo create a new file /etc/nginx/sites-enabled/my-super-api.saxion.online.conf and give it the following contents:

```
server {
    server_name my-super-api.saxion.online;
    listen 80;
    location / {
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header Host $host;

            proxy_pass http://localhost:3000/;

            proxy_buffering off;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
    }
}
```

Use your own domain name here

Restore the contents of the default config file:

```
location / {
   try_files $uri $uri/ =404;
}
```

Test the nginx config:

```
> sudo nginx -t
```

And restart nginx:

```
> sudo systemctl restart nginx
```

Loading the domain in the browser should still show the hi-service.

Create *Let's Encrypt* TLS certificates

Creating and configuring the TLS certificates has been totally automated for the configuration we now have when using certbot.

Install the Let's Encrypt certbot for nginx:

```
> sudo apt install certbot python3-certbot-nginx -y
```

Start the certbot configuration wizard:

```
> sudo certbot --nginx -d my-super-api.saxion.online
```

- Fill out your emailaddress
- Accept the terms
- Accept or reject spam

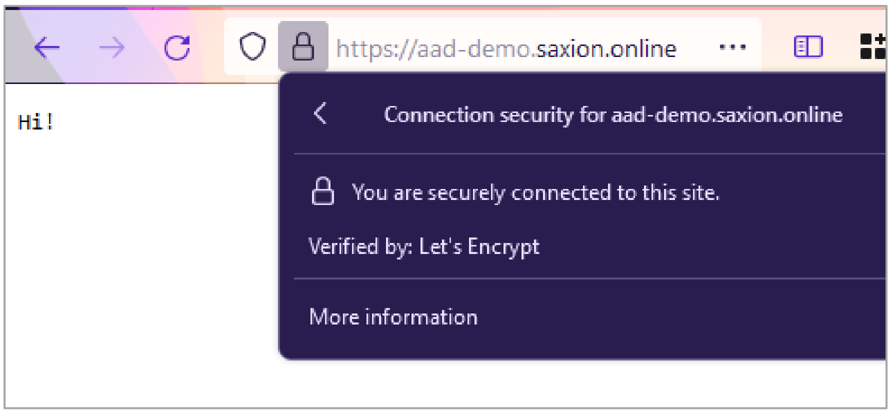
It will now create the certificates and add them to the nginx config, in my case "aad-demo.saxion.online":

```
Requesting a certificate for aad-demo.saxion.online

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/aad-demo.saxion.online/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/aad-demo.saxion.online/privkey.pem
This certificate expires on 2023-12-14.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.

Deploying certificate
Successfully deployed certificate for aad-demo.saxion.online to /etc/nginx/sites-enabled/aad-demo.saxion.online.conf
Congratulations! You have successfully enabled HTTPS on https://aad-demo.saxion.online
```

Now the website should be reachable by https. Well done!


https://aad-demo.saxion.online
Hi!
Connection security for aad-demo.saxion.online
You are securely connected to this site.
Verified by: Let's Encrypt
More information

Appendix: create a service for a docker container

A docker container will go down and not automatically restart again whenever your linux server restarts.

To automatically restart a docker container, you need to define a systemd config file and let systemctl manage the service.

First sudo create a file (using vi or nano) at the location /etc/systemd/my-service-name.service, where you change my-service-name with your own service name.

The contents should be something like this:

```
[Unit]

Description=My beautiful API container service
Requires=docker.service
After=docker.service

[Service]
Restart=always
ExecStart=/usr/bin/docker run -p 3000:3000 --name aad-demo
    your-docker-username/aad-demo

[Install]
WantedBy=default.target
```

This should be one line, containing your own docker run command.

After this, register the service with systemctl:

```
> sudo systemctl enable /etc/systemd/my-service-name.service
```

And start it:

```
> sudo systemctl start my-service-name
```

It should now restart automatically whenever the server is restarted.