Cloudflare CSE Technical Project

Done by: Lim Yin Shan

Prepared for: An Qing Jiao, Gracelin Chan and the team at Cloudflare

**This report aims to:**

1. Guide you in how I created the working application, step-by-step
2. How to access the working application
3. Provide the link between the step/product to a possible use case
4. Provide rationales as to why certain services/configuration were used
5. Address the problems both technically and non-technically

I would like to emphasise that prior to this project, my experience and knowledge with the whole internet architecture and Cloudflare services/product are little to none.

Hence, please let me know if there are concepts which I misunderstood, configurations that I have set wrongly or improvements that can be made.

# Table of Contents

# The Internet Architecture

Before I deep dive into the application itself, it is important to understand how the internet works before one can comprehend what each step in the project document refers to. As the project focuses a lot on the Domain Name System (DNS) of the internet architecture, this section would cover mostly about the DNS.

## Terminologies: DNS, IPs, Domains, Name servers and more

|  |  |  |
| --- | --- | --- |
| Terminology | Description | Analogy |
| DNS | The overall system that translates a human-readable domain to a computer-friendly IP | The entire Google map system |
| Internet Protocol (IP) address | A unique, computer-readable address of a device that is connected to the internet  e.g 44.203.2.85 | Longitude and Latitude of a home address  e.g 1.2797233638774987, 103.84752194602784 |
| Domains | Human-readable address  e.g www.limyinshan.com | Home address  e.g 182 Cecil St, #35-01 Frasers Tower, Singapore 069547 |
| Name Servers | Contains the DNS records and handles the translation of domain into IPs | Google Maps’ servers |
| DNS records | Stores information associated with domains including how traffic should be routed | Individual map entry that store information linking a home address to its latitude |

Figure 1: Table containing main Internet terminologies

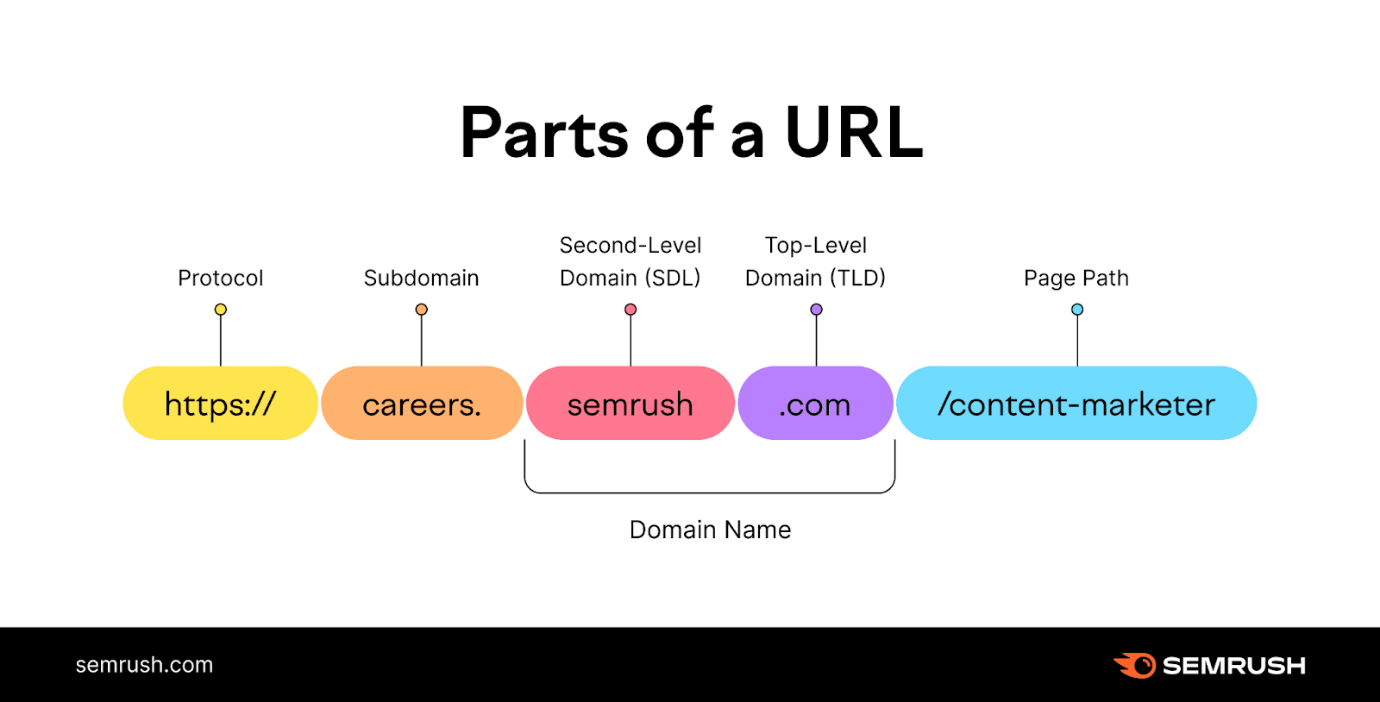


Figure 2: Understanding a URL

## The DNS Process

1. When you enter a URL (hostname) in your web browser such as Google Chrome, it will make a DNS query to find out the unique IP address that is associated with the hostname
2. It will start by looking at your local browser cache (your browser memory), if you have previously entered and visited the respective hostname you entered before
3. If your cache does not contain the information, it would then look up on the various name servers via the DNS recursive resolver
   1. First it would make a request to the root name server, which would give back a response of the respective TLD name server of the hostname e.g “.com name server”
   2. Then it would make a request to the “.com” TLD name server and return a response of the respective authoritative name server
   3. Then, it would make a last request to the authoritative name server, and this authoritative name server would usually contain the exact DNS records that contains information of the respective IP address to the entered hostname
   4. The IP address is then sent back to the client and its cache (for future use)
4. Now knowing the IP address, the browser can then send a request to the server and the server will send back a response to the browser
5. The browser/client then renders the response accordingly, such that you will be able to view the content on your screen

# Project Pre-Requisites

With a better understanding of how DNS works, we can now dive into how the application is being created. But before that, there are a few pre-requisites.

1. Domain: Creating/Registering for a domain name
2. Cloudflare: Creating a Cloudflare account and linking the domain to Cloudflare’s name servers

## Domain

Before diving into the project steps, a domain name is registered first such that it can be link to the server’s IP address later on. There are many reasons as to why a domain name is preferred over the use of IP addresses, but these are the main ones:

1. Human-readability – domain names are easier for humans to remember and identify as they are made up of recognisable words
2. Flexibility – if the IP address of a server were to change (and this happens relatively frequent), it is very easy to route the domain name to the new IP without much downtime or impact to services
3. Scalability – you can associate more than 1 IP address, subdomains and more with a single domain name, allowing you to manage many resources with a single identify

To register for a domain name, there are certain factors to consider such as domain name provider and costs. However, in the context of this project, as there would definitely be certain Cloudflare service configurations, I registered the domain under Cloudflare as there are no free alternatives currently and it automatically links the Cloudflare’s name servers to my domain. By linking to Cloudflare’s name servers, it would mean routing my website’s traffic through Cloudflare’s network and delegating the management of my domain’s DNS records to Cloudflare. This link would allow Cloudflare to handle the whole DNS resolution process, improve security, website performance and so much more.

* Domain name: **limyinshan.com**

# Breakdown and Summary of the Steps

|  |  |  |
| --- | --- | --- |
| # | Step detail | Step’s Goal, Product, and Use Cases |
| 1 | Create an origin web server on a platform of your choosing. This could be in AWS, Google  Cloud, DigitalOcean, your Raspberry Pi, etc.  This web server must run an endpoint that returns all HTTP request headers in the body of the HTTP response.  The web server can be something that you have written yourself (e.g. in JavaScript, Python, etc) or by using a 3rd party application such as HTTPBin. Proxy traffic to this server through Cloudflare. | Creating a web server and proxying the traffic to the server through Cloudflare by:   1. Creating the web server 2. Creating the respective DNS record (A record) 3. Creating the script to return HTTP request headers   The main point for this step is to proxy traffic through Cloudflare so that the user can utilise Cloudflare’s services in DNS management, etc. |
| 2 | Secure the communication between Cloudflare and your Origin Server with a non-Cloudflare provisioned TLS certificate using at least the Full-Strict mode on Cloudflare. | 1. Creating a third-party, trusted CA certificate 2. Ensuring SSL/TLS encryption mode is Full (strict)   The main point for this step is to ensure that there is full, end-to-end encryption to prevent data tampering and theft.  Illustrate what TLS and SSL for and their differences |
| 3 | Install and configure Cloudflare Tunnel on your origin server using a subdomain called  “tunnel”, e.g. tunnel.yourwebsite.com. Make connections proxied to your server protected  using this tunnel. | 1. Creating a Cloudflare tunnel 2. Creating the respective DNS record (CNAME record)   The main point or use case of this step is to create a secure way to connect your own resources to the internet without a public routable IP, via Cloudflare.  Demonstrate differences between tunnel and VPN |
| 4 | Write an API call that outputs all of your DNS records, using an API scoped token. Include the token permission scope, API call and its output. | 1. Create a script that can call Cloudflare API   The main purpose for this step is to experiment using Cloudflare’s API with scoped permissions. |
| 5 | Create a Cloudflare Worker. The HTTP response body should be “This is your ${CLIENT\_IP}  and you are accessing this site from ${COUNTRY) | ${ASN}.  a. This response should be visible on the browser as HTML. Run this worker script on  the /geo path. For example: www.yourwebsite.com/geo.  b. Use Workers to create a logic if a user who is not from Singapore to be redirected to  https://1.1.1.1/.  c. Create this worker using the Wrangler CLI, upload your Workers code to a public Git  repository for your implementation. | 1. Use Wrangler CLI to create a Cloudflare Worker 2. Create a simple response body script and redirector 3. Deploy Cloudflare Worker   The main use case and purpose of this step is to experiment using a CLI tool like wrangler to configure Cloudflare services and using Workers to deploy code quickly, anywhere. |
| 6 | Lock down access for a particular path for your Cloudflare Tunnel subdomain (e.g.  tunnel.yourwebsite.com/secure) and only allow access for a particular user or a group of users using Cloudflare Zero Trust.  a. Ensure nobody can bypass Cloudflare and access your server’s IP directly. | 1. Use Cloudflare Zero Trust System to restrict access to paths 2. Create firewall rules to prevent direct server IP access   The main purpose of this step is to experiment using Cloudflare’s Zero Trust System to implement different ways to restrict/allow access to certain paths and protecting unauthorised access directly via the server’s IP. |

# DNS records – Adding A, AAAA, or CNAME records

* A record – for web server that has IPv4 address
* AAAA record – for web server that has IPv6 address
* CNAME record – Canonical Name record, when you want to alias one domain or subdomain to another domain
* MX record – *I don’t think I’m using this*
* Reason for adding these DNS records:
  + For “www” – allows visitors to access your website by typing "**www.**limyinshan.com" in their browsers (the DNS record specifies the destination for the "www" subdomain, ensuring that it resolves correctly)
  + For root domain “**limyinshan.com**”—adding a DNS record for your root domain to resolve to an IP address or CNAME, ensuring that users can reach your website by entering just the domain name without the "www" prefix

# Step 1’s process (linux instance)

1. create domain name through cloudflare

2. create ec2 server instance on AWS

3. ssh into the ec2 server and install nginx web server and run: `sudo amazon-linux-extras install nginx1` and `sudo service nginx start` and `systemctl status nginx`

4. created script (flask app) to return all HTTP request headers

5. install required python modules: `pip3 install flask`

6. add A record for my domain so that www.limyinshan.com points to 50.17.53.22 (my ec2 public ip) and has its traffic proxied through Cloudflare

6. modify nginx.conf to listen to port 80 and reverse proxy to port 5000 (since flask app will run on port 5000): `sudo nano /etc/nginx/nginx.conf`

7. run the flask app: `python3 app.py` (if permission denied for flask app to run, chmod 707 first)

------8. update inbound rule of security group of ec2 server to accept tcp requests through port 5000 from anywhere------

------9. use web browser to check whether flask app running well and endpoint works aka http://50.17.53.22:5000/ returns a response body containing header requests-----

8. use web browser to check whether flask app running well and endpoitns works via https://www.limyinshan.com/, should return response body containing header requests

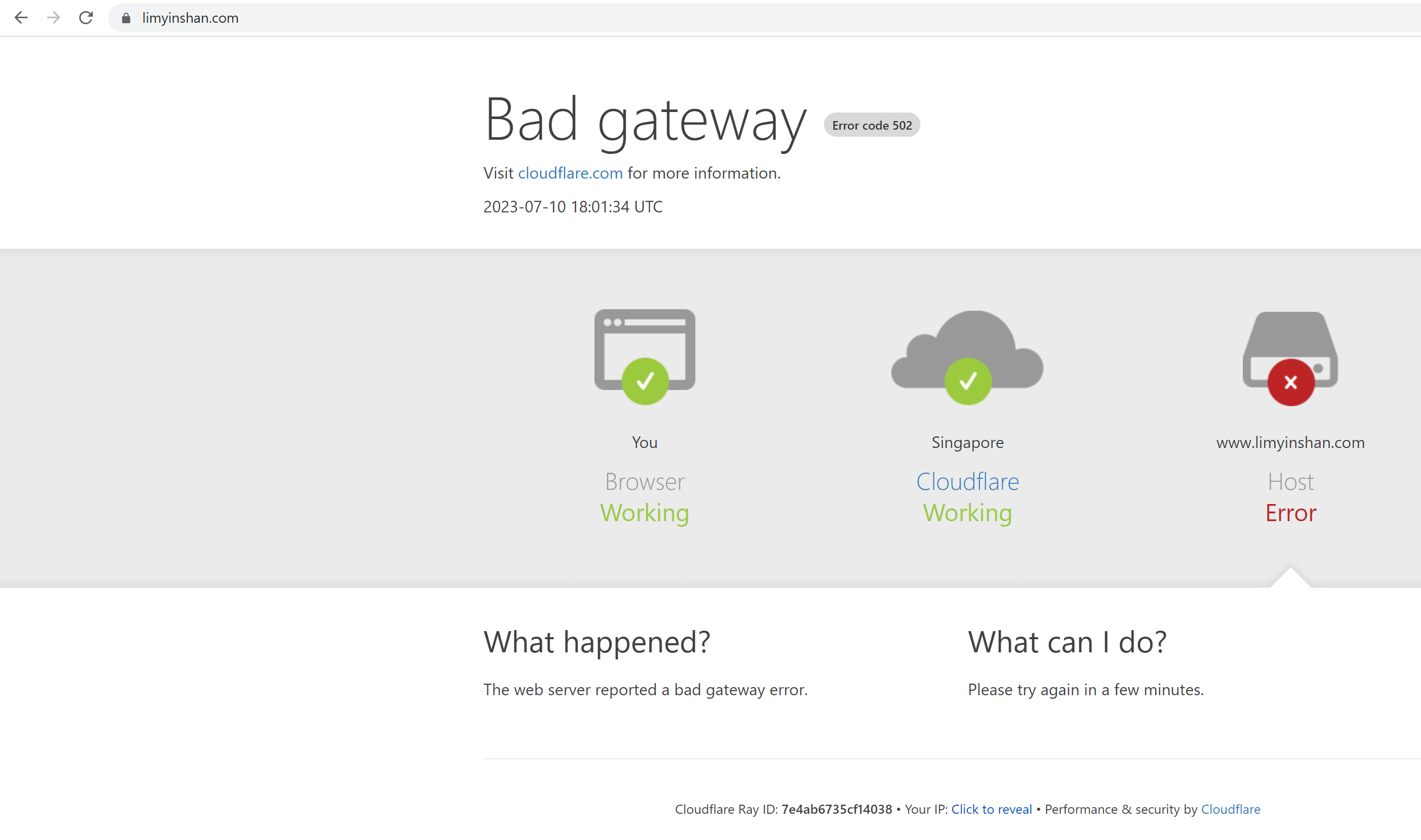
9. to ensure that flask app runs even after i exited the ec2 instance, instead of `python3 app.py`, use `nohup python3 app.py &` and check with `pgrep python3`

-> & allows us to run the application in background and nohup allows the application to keep running even on hang up/logout

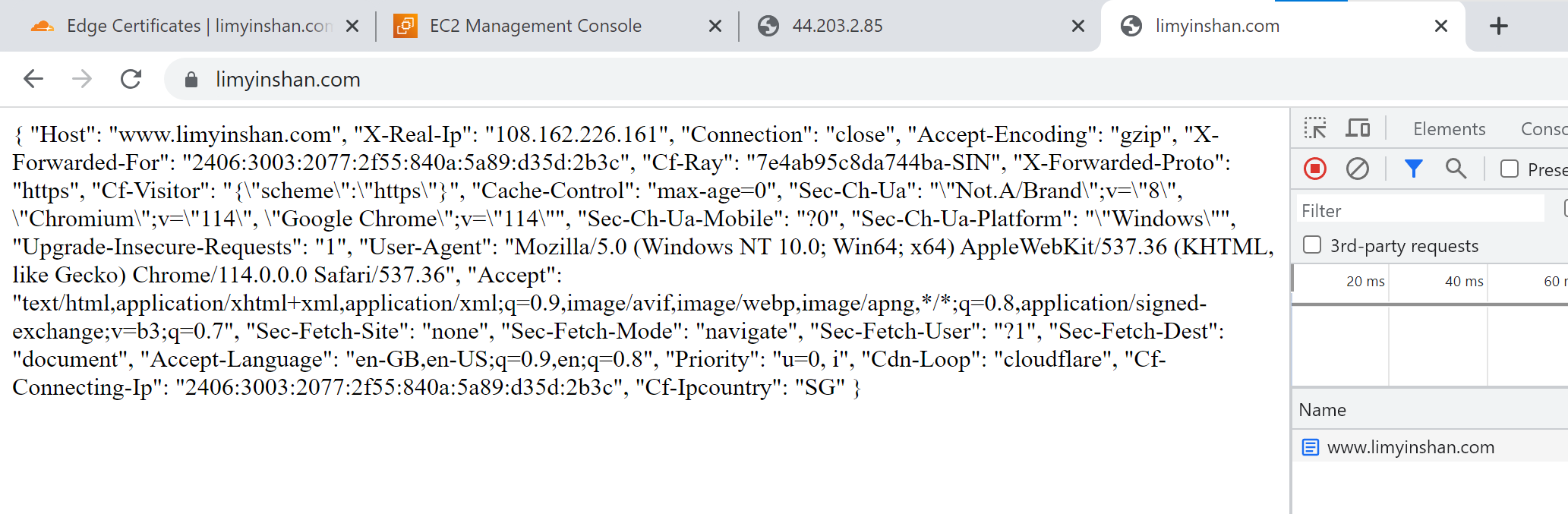
-> `sudo kill <process-id>` to kill the nohup process

# Step 1’s process (ubuntu instance)

1. create domain name through cloudflare
2. create ec2 server instance on AWS
3. ssh into the ec2 server and install nginx web server and run: `sudo apt-get install nginx` and `sudo systemctl status nginx`
4. created script (flask app) to return all HTTP request headers
5. install pip modules: `sudo apt install python3-pip`
6. install required python modules: `pip3 install flask`
7. add A record for my domain so that www.limyinshan.com points to 50.17.53.22 (my ec2 public ip) and has its traffic proxied through Cloudflare
8. modify nginx.conf to listen to port 80 and reverse proxy to port 5000 (since flask app will run on port 5000): `sudo nano /etc/nginx/nginx.conf`
9. test syntax and restart nginx server: ` sudo nginx -t` and `sudo service nginx restart`
10. run the flask app: `python3 app.py` (if permission denied for flask app to run, chmod 707 first)
11. use web browser to check whether flask app running well and endpoints works via https://www.limyinshan.com/, should return response body containing header requests
12. to ensure that flask app runs even after i exited the ec2 instance, instead of `python3 app.py`, use `nohup python3 app.py &` and check with `pgrep python3`
    1. & allows us to run the application in background and nohup allows the application to keep running even on hang up/logout
    2. `sudo kill <process-id>` to kill the nohup process



Shows that successfully proxy through cloudflare (test by not running flask app)



After running flask app

# Step 1’s end result and benefit

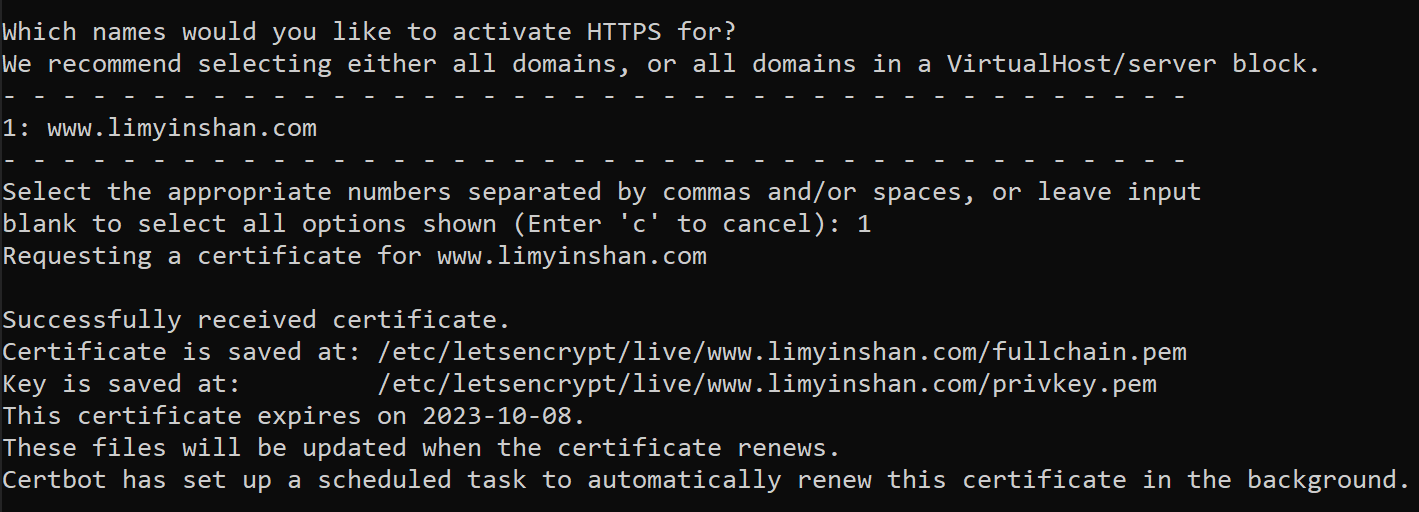
Proxying through Cloudflare refers to the practice of routing incoming web traffic through Cloudflare's network before reaching your origin server (in this case, your EC2 instance). Cloudflare acts as an intermediary or reverse proxy between the client (visitor) and your origin server. Here's what proxying through Cloudflare entails and the benefits it provides:

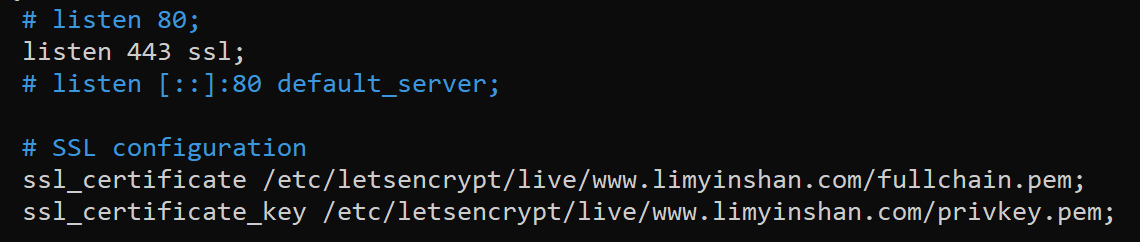
1. Traffic distribution and load balancing:
   1. When you enable Cloudflare proxy for your domain, incoming requests are automatically distributed across Cloudflare's global network of data centers. This helps distribute the load and handle traffic spikes more efficiently. Cloudflare's intelligent routing algorithm directs each request to the nearest data center, reducing latency and improving response times.
2. DDoS protection and security:
   1. Cloudflare provides robust protection against Distributed Denial of Service (DDoS) attacks by filtering and absorbing malicious traffic before it reaches your origin server. Cloudflare's DDoS protection helps ensure your website remains available and accessible during such attacks.
   2. Cloudflare also offers various security features such as web application firewall (WAF), SSL/TLS encryption, IP reputation-based blocking, and bot mitigation, which enhance the security and integrity of your website.
3. Content caching and optimization:
   1. Cloudflare caches static content from your website on its edge servers, closer to the end-users. This reduces the load on your origin server and improves content delivery speed. Cached content is served directly from the nearest Cloudflare edge server, reducing latency and improving overall website performance.
4. SSL/TLS termination:
   1. Cloudflare acts as a SSL/TLS termination point, allowing you to secure your website with HTTPS even if your origin server doesn't have an SSL certificate. Cloudflare handles the encryption and decryption of HTTPS traffic between the client and its edge servers, providing a secure connection.
5. Analytics and insights:
   1. Cloudflare provides detailed analytics and insights about your website's traffic, including visitor statistics, bandwidth usage, and security events. You can gain valuable insights into your website's performance and security through the Cloudflare dashboard.

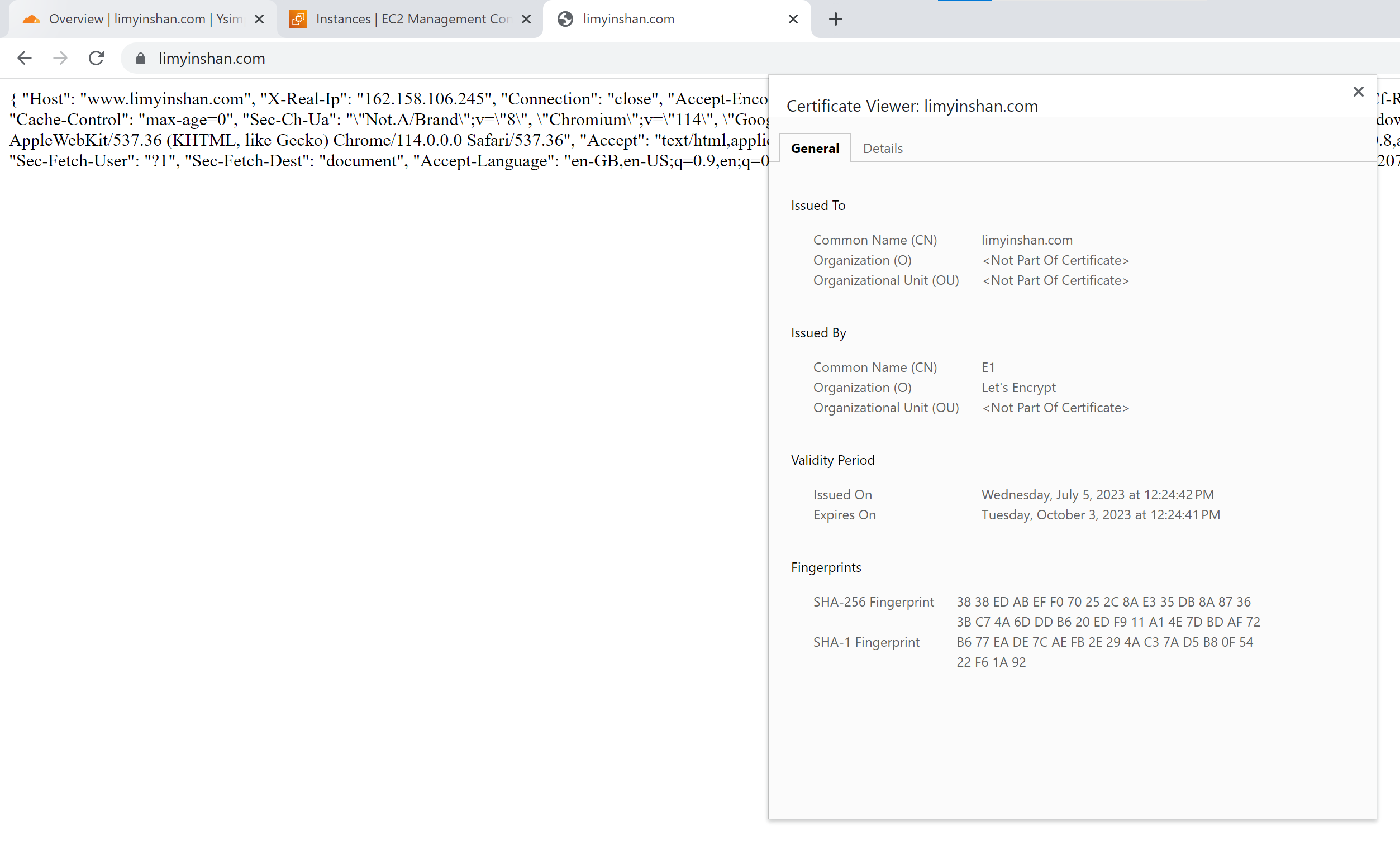
By proxying your traffic through Cloudflare, you can leverage these benefits to enhance the performance, security, and availability of your website or web application. Cloudflare's network and services help optimize content delivery, protect against threats, and provide valuable insights to improve the overall user experience.

# Step 2

1. Choose a certificate authority (CA):
   1. Research and choose a trusted CA from which you would like to obtain the TLS certificate. Some popular options include Let's Encrypt, Sectigo (formerly Comodo), Digicert, and GlobalSign.
   2. Consider factors such as certificate cost, validity period, CA reputation, and compatibility with your server software.Install snapd for certbot installation (for TLS cert): `sudo apt install snapd`
   3. Chose Let’s Encrypt as the trusted CA for TLS certificate provider <https://letsencrypt.org/getting-started/> due to ease of installation/renewal via certbot and trustworthiness and credibility
2. Install certbot: `sudo snap install --classic certbot`
3. Run Certbot to obtain and install the TLS certificate for your domain
   1. `sudo certbot certonly --nginx --preferred-challenges http-01`
   2. Certonly command ensures that Certbot only obtains the certificate without modifying your Nginx or Cloudflare configurations
   3. Follow the respective instructions that appear

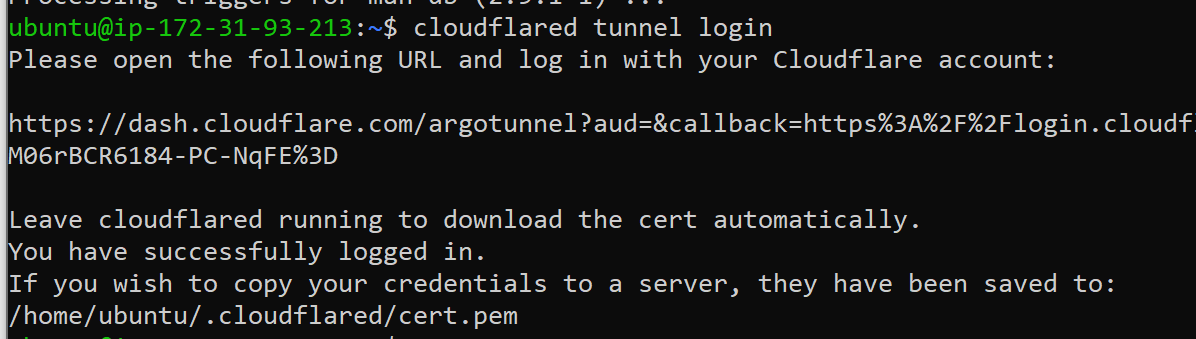
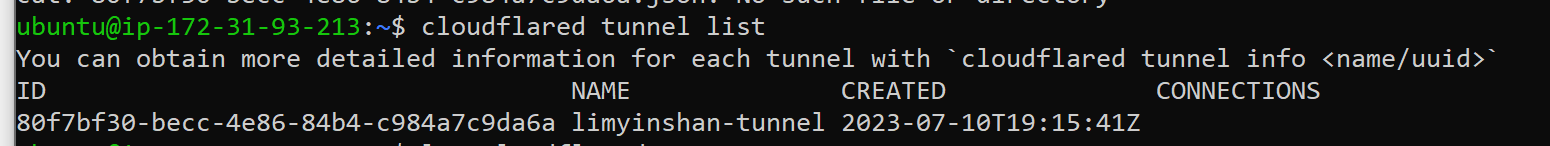
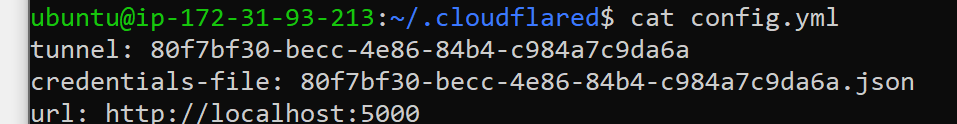
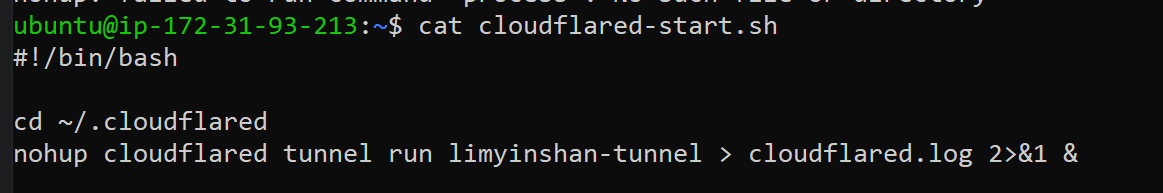


1. Update nginx conf with new certificates and restart
   1. 
   2. `sudo nginx -t`
   3. `sudo systemctl restart nginx`
2. Certbot has set up automatic renewal for your certificate, so it will be automatically renewed before it expires
3. Enable full (strict) ssl/tls encryption mode on cloudflare dashboard after certificate created on server!

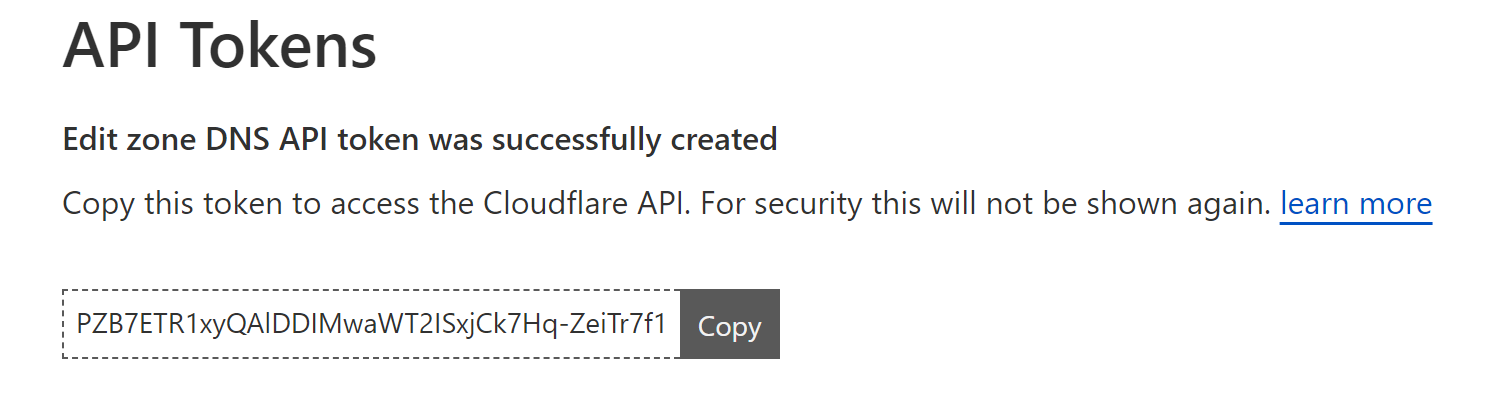


# Step 3

<https://developers.cloudflare.com/cloudflare-one/connections/connect-apps/install-and-setup/tunnel-guide/local/>

1. Install the Cloudflare Tunnel CLI: `wget -q https://github.com/cloudflare/cloudflared/releases/latest/download/cloudflared-linux-amd64.deb && sudo dpkg -i cloudflared-linux-amd64.deb`
2. authenticate the Cloudflare Tunnel CLI: `cloudflared tunnel login` and follow the steps
   1. 
3. Create a cloudflare tunnel: `cloudflared tunnel create limyinshan-tunnel`
4. Verify that it is created using: `cloudflared tunnel list`
   1. 
5. Create a config file to route traffic from the origin server to the hostname tunnel as per the step instruction
   1. 
6. Assign a CNAME record so that it can point traffic to the newly created tunnel subdomain: `cloudflared tunnel route dns limyinshan-tunnel tunnel.limyinshan.com`
7. Run the tunnel to proxy incoming traffic to origin server: `cloudflared tunnel run limyinshan-tunnel`
8. To check tunnel info: `cloudflared tunnel info limyinshan-tunnel`
9. To run the tunnel permanently:
   1. Create a shell script that enters cloudflare working directory and run the command
   2. 

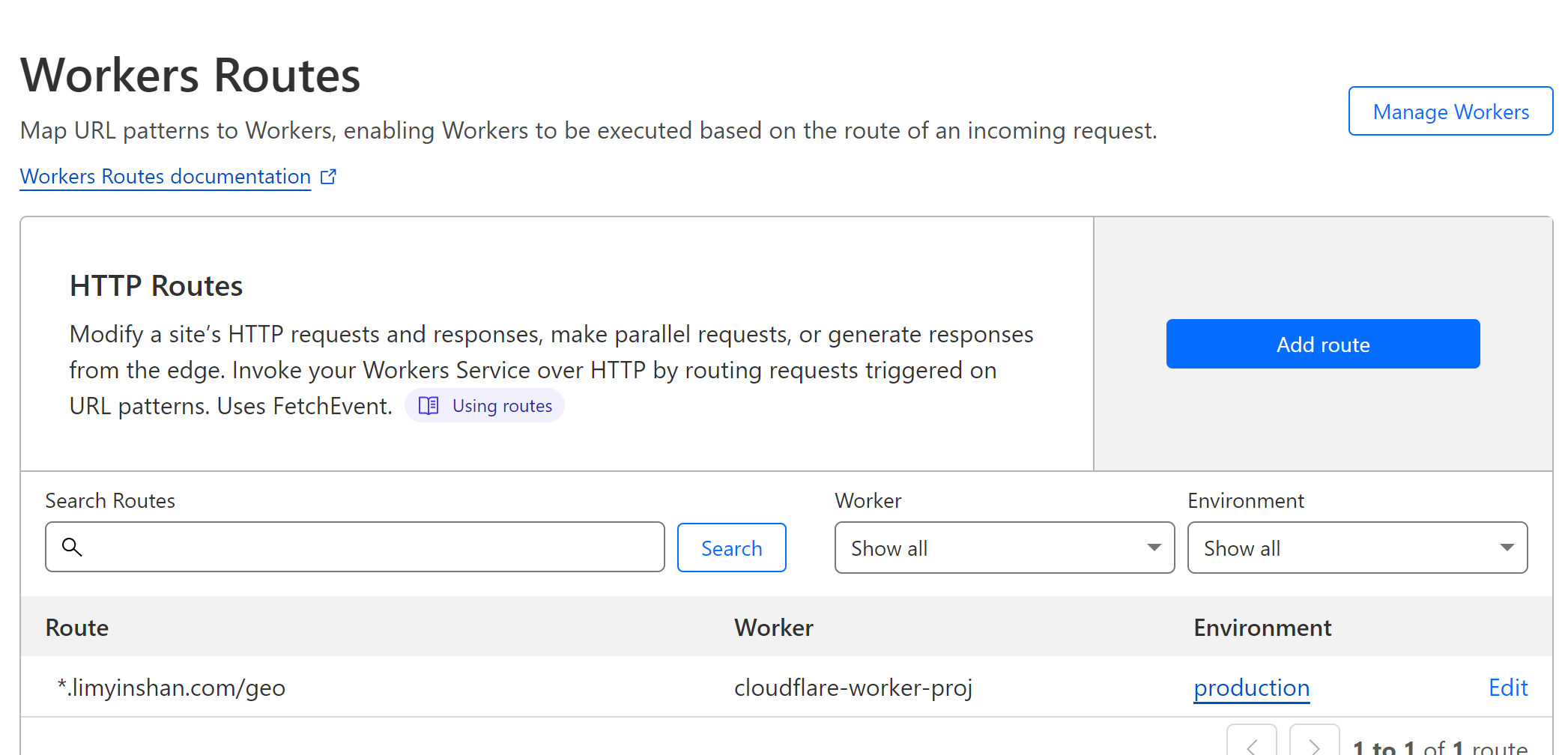
# Step 4

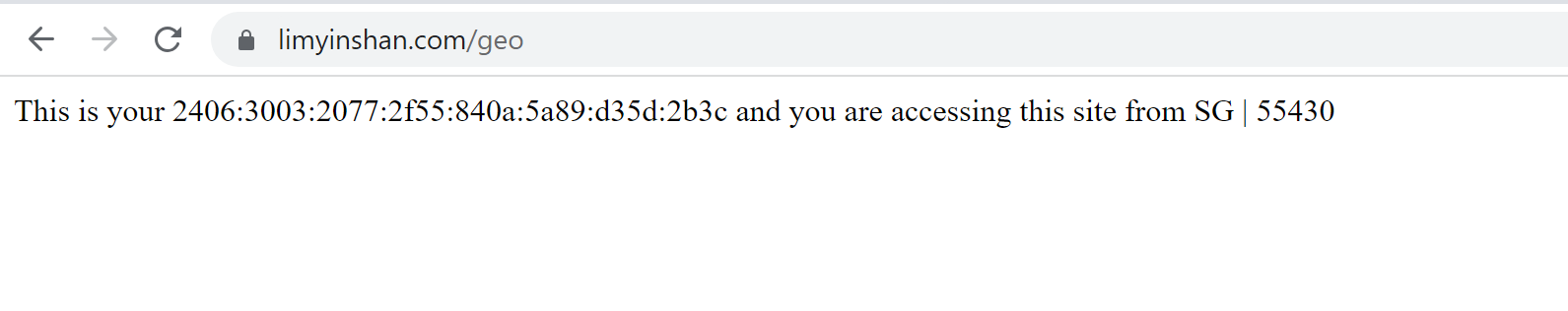
1. Create an API scoped token just for retrieving the DNS records
   1. Cloudflare dashboard > My profile > API Tokens > Create token
   2. Use “Edit Zone DNS” template
   3. Allow Read permissions
   4. Only include the zone for the respective domain 🡪 limyinshan.com **(use all zones instead if you want to see DNS records from all domains)**
2. Once API scoped token is created, you should be able to copy the token
   1. 

## API Call that outputs all of DNS records for limyinshan.com

|  |  |
| --- | --- |
| **API Call** | **Params** |
| curl -X GET "https://api.cloudflare.com/client/v4/zones/{*zone\_id*}/dns\_records" \  -H "Authorization: Bearer {*Token*}" \  -H "Content-Type: application/json" | * *Zone\_id* – zone id for the domain limyinshan.com   + Get the respective zone\_id by **excluding** the paths after “zones” in the function call   + E.g GET “https://api.cloudflare.com/client/v4/zones” * *Token* – API scoped bearer token   **Scopes added:**   1. Zone.DNS permission 2. Read permission 3. 1 Zone (limyinshan.com)   **Example:**  curl -X GET "https://api.cloudflare.com/client/v4/zones/20a7b4184301472531b4b6cb08723ee0/dns\_records" \  -H "Authorization: Bearer PZB7ETR1xyQAlDDIMwaWT2ISxjCk7Hq-ZeiTr7f1" \  -H "Content-Type: application/json"  **Response/Output:**  {  "result": [  {  "id": "492b2143acf41e6e2e5c09f0aef5674b",  "zone\_id": "20a7b4184301472531b4b6cb08723ee0",  "zone\_name": "limyinshan.com",  "name": "www.limyinshan.com",  "type": "A",  "content": "44.203.2.85",  "proxiable": true,  "proxied": true,  "ttl": 1,  "locked": false,  "meta": {  "auto\_added": false,  "managed\_by\_apps": false,  "managed\_by\_argo\_tunnel": false,  "source": "primary"  },  "comment": "www.limyinshan.com points to 44.203.2.85 and has its traffic proxied through Cloudflare.",  "tags": [],  "created\_on": "2023-07-10T15:05:53.365914Z",  "modified\_on": "2023-07-10T17:41:09.939064Z"  },  {  "id": "db2b4a5be794ef7f97da6007ac8bf590",  "zone\_id": "20a7b4184301472531b4b6cb08723ee0",  "zone\_name": "limyinshan.com",  "name": "tunnel.limyinshan.com",  "type": "CNAME",  "content": "80f7bf30-becc-4e86-84b4-c984a7c9da6a.cfargotunnel.com",  "proxiable": true,  "proxied": true,  "ttl": 1,  "locked": false,  "meta": {  "auto\_added": false,  "managed\_by\_apps": false,  "managed\_by\_argo\_tunnel": false,  "source": "primary"  },  "comment": null,  "tags": [],  "created\_on": "2023-07-10T20:16:32.153796Z",  "modified\_on": "2023-07-10T20:16:32.153796Z"  }  ],  "success": true,  "errors": [],  "messages": [],  "result\_info": {  "page": 1,  "per\_page": 100,  "count": 2,  "total\_count": 2,  "total\_pages": 1  }  } |

# Step 5

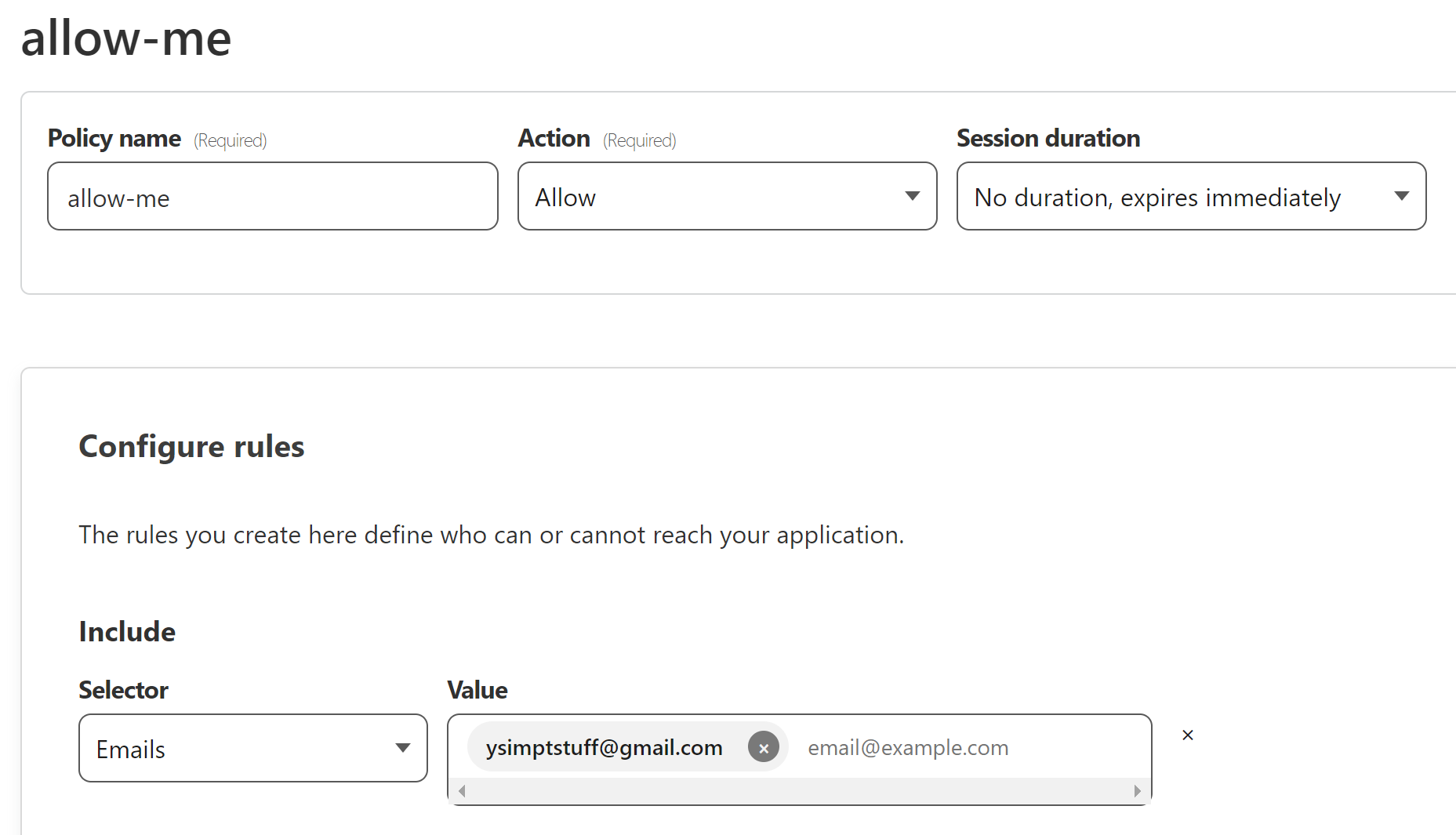
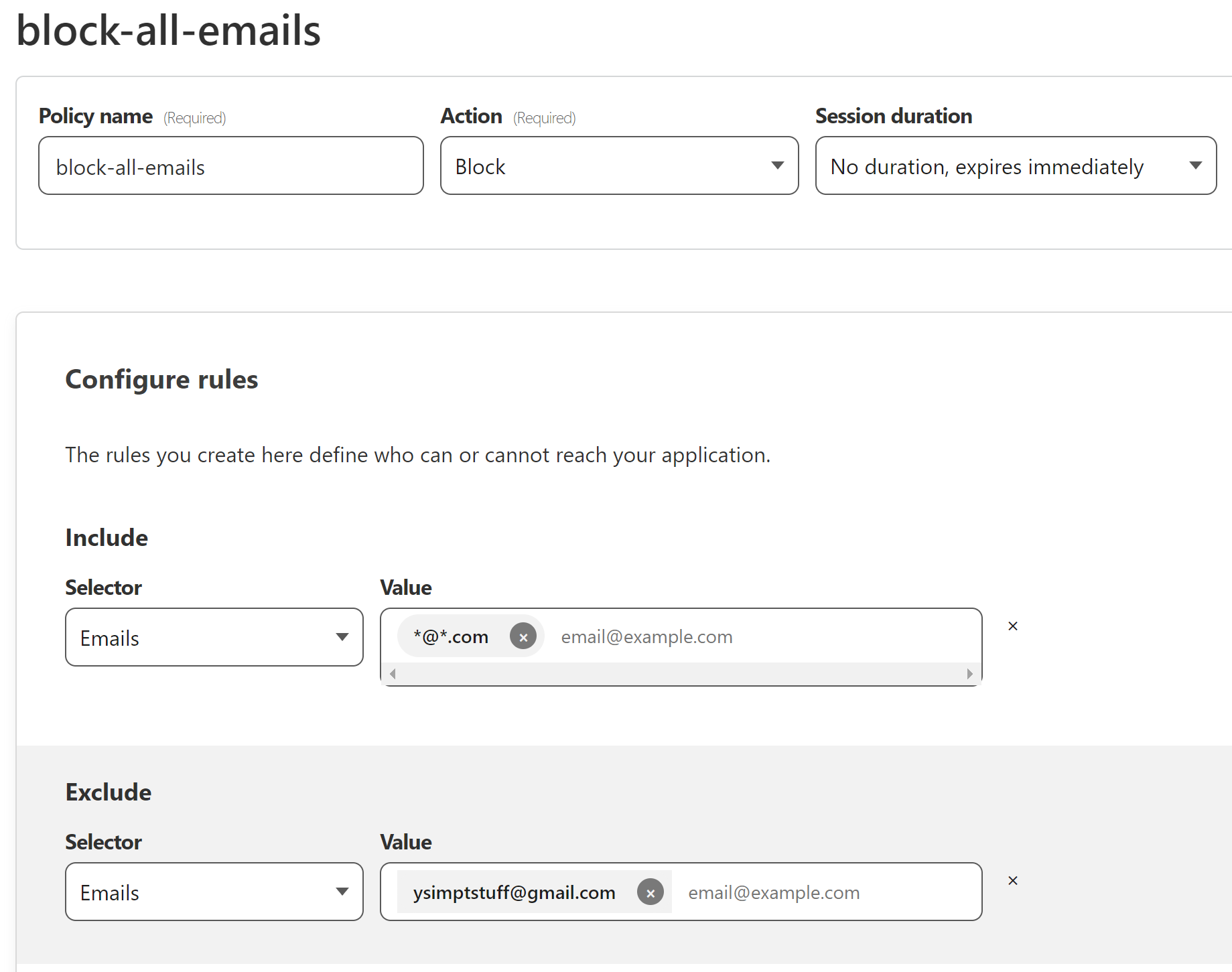
1. Install wrangler: `npm install wrangler -g` (assumes you already have npm and node installed and node version of 16.13.0 or later)
2. Create cloudflare worker via CLI: `wrangler init cloudflare-worker-proj` and follow the instructions accordingly
   1. Choose the option "Hello World" Worker as you only need a simple base to implement the required functionality
   2. Do not deploy yet as there are still some configurations to complete
3. Update src/worker.js file according to the required response body
4. Update wrangler.toml file to tell the configurations for worker creation
5. Deploy worker to cloudflare using: `wrangler deploy`
   1. Very fast in updating the js file, changes are reflected on “/geo” route really quickly
6. After creating the worker via Wrangler CLI, create and configure the route for “/geo” path
   1. 

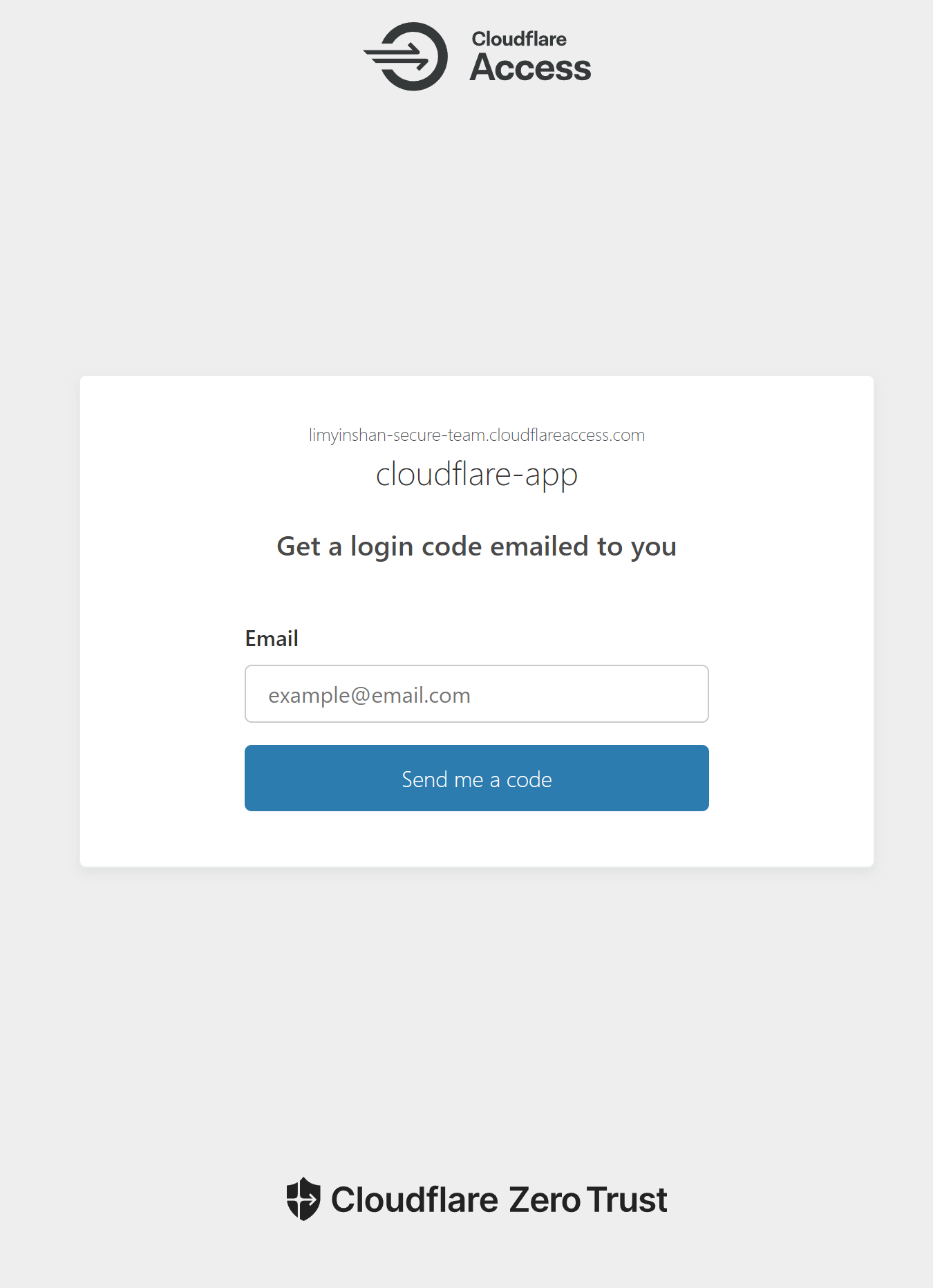


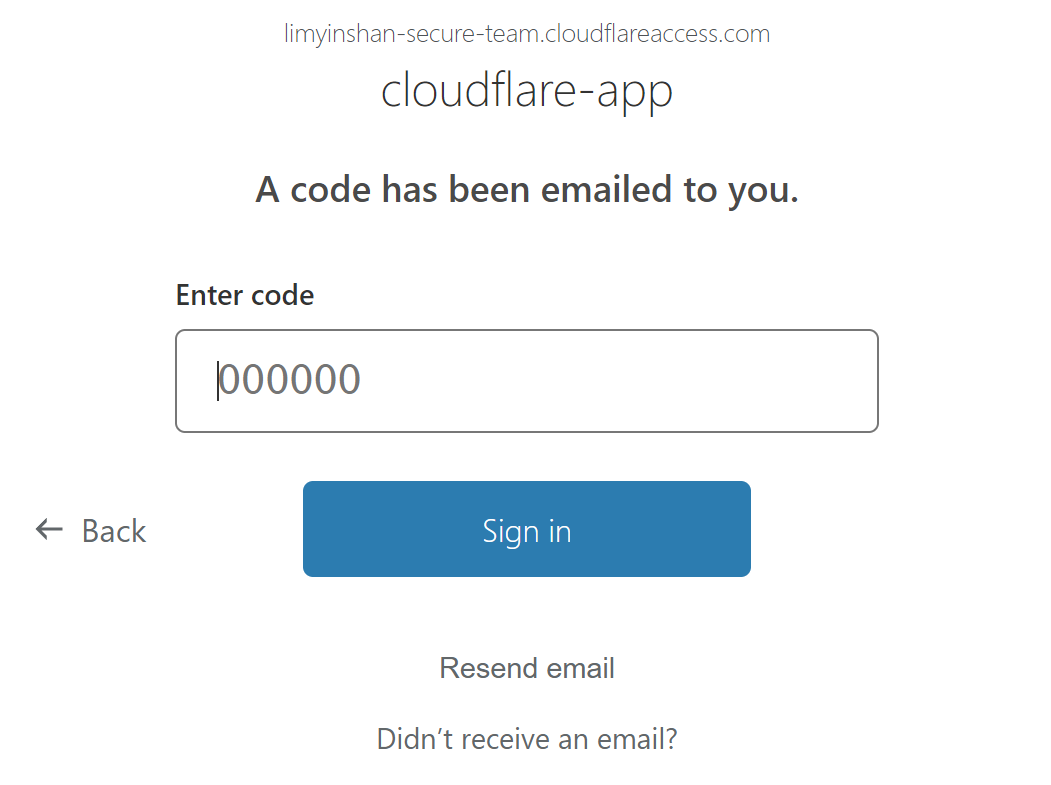
Step 5 (Workers) Benefits for use

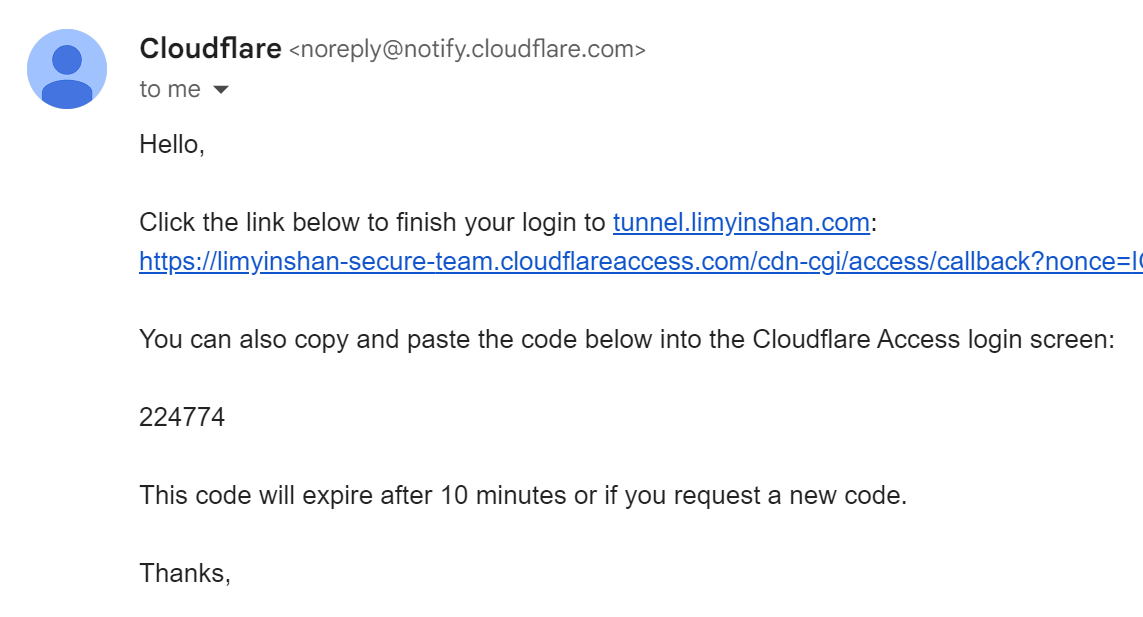
1. (to edit) very quick upload js and reflect on web server already because worker upload to servers cache all around the world 🡪 etcetc

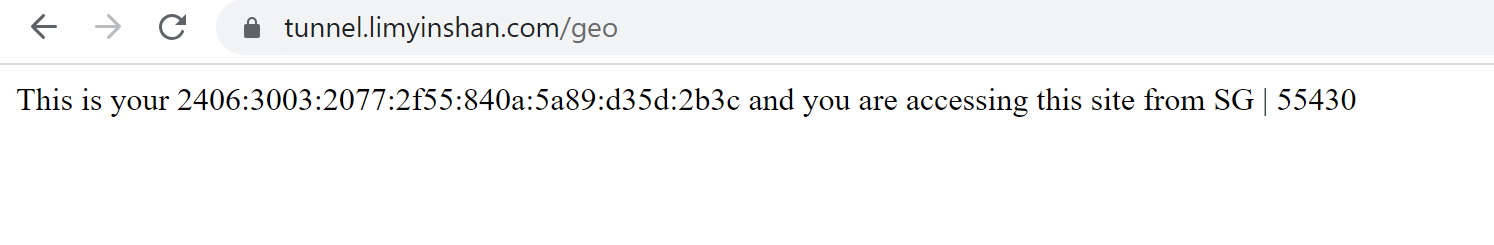
# Step 6

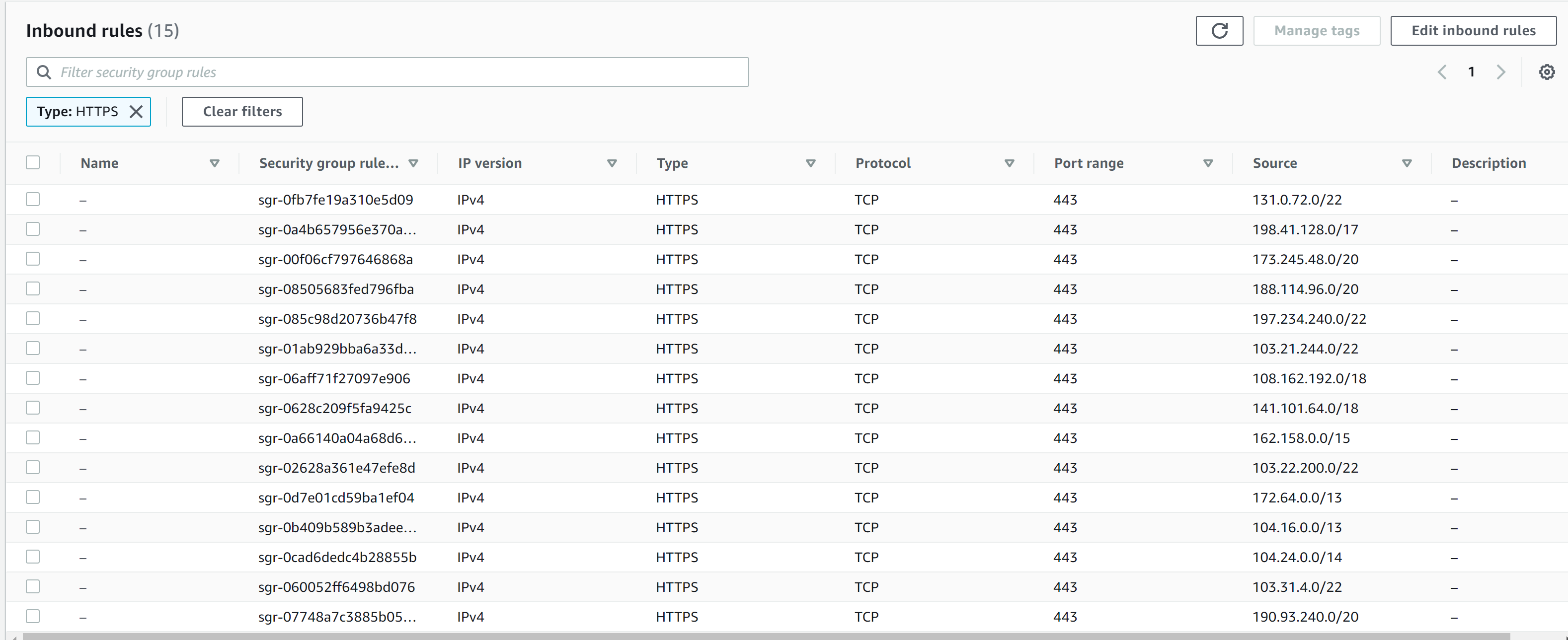
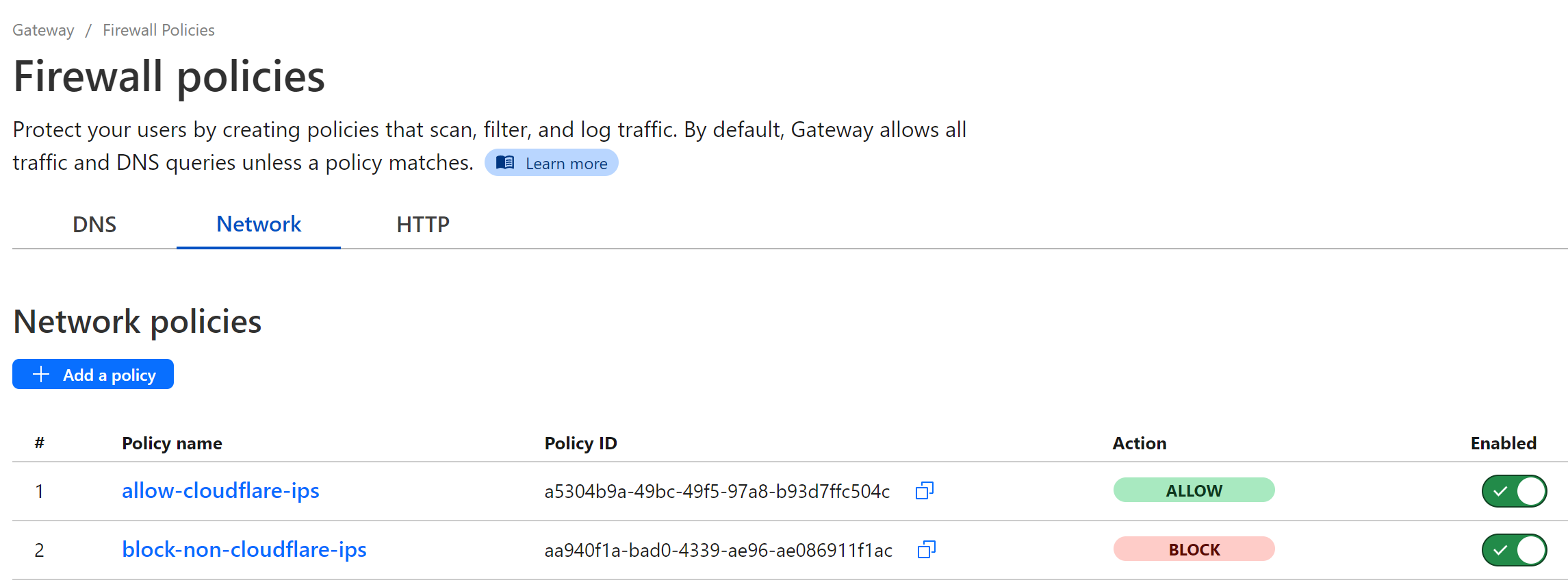
1. Head over the cloudflare’s Zero Trust dashboard
2. Create an application for restricted access to the “/geo” path for the tunnel subdomain
3. Set these configurations for the application:
   1. session duration as “no duration, expires immediately” for debugging purposes
   2. subdomain as tunnel
   3. domain as limyinshan.com
   4. path as geo
   5. Ensure “Enable App in App Launcher” is checked and using default domain
   6. Create two main policies, one to reject all emails, and one to only accept a particular user or a group of user emails (for this case, I will only be using my email)
      1. 
      2. 
      3. This is ensure that only the email [ysimptstuff@gmail.com](mailto:ysimptstuff@gmail.com) is able to access tunnel.limyinshan.com/geo.
      4. One thing to note would be that if you were to key in an email that is not the allowed email, you will not be receiving any code nor prompt saying that your email is not part of the allowed list
4. After creation of the app and its policies, head over to tunnel.limyinshan.com/geo and it will redirect you to a Cloudflare Access site <https://limyinshan-secure-team.cloudflareaccess.com/> that prompts you to enter an email where you will receive a verification code if your email is part of the allowed list









1. To further ensure nobody can bypass Cloudflare and access server’s IP directly, there are firewall rules in place to only accept source IPs from Cloudfare IP ranges, on both Cloudflare and AWS servers.
   1. 
   2. 

Future Improvements

1. Improvements **on server side**
   1. Elastic ip
   2. Autoscaling servers tied to ELB