Cloudflare CSE Project Report

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 and what is the expected result
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

Deliverables included:

1. This report
2. GitHub repository containing various code configurations, scripts and others
   a. https://github.com/jenniupdates/cloudflare-technical-project
3. Domain: www.limyinshan.com
4. Tunnel: https://tunnel.limyinshan.com
5. Secured Path (Zero Trust System): https://tunnel.limyinshan.com/geo

This report was written to also cater for readers who may not have a strong technical background. As a result, there may be additional explanations provided to ensure better understanding of the concepts and steps involved such that most can follow.

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

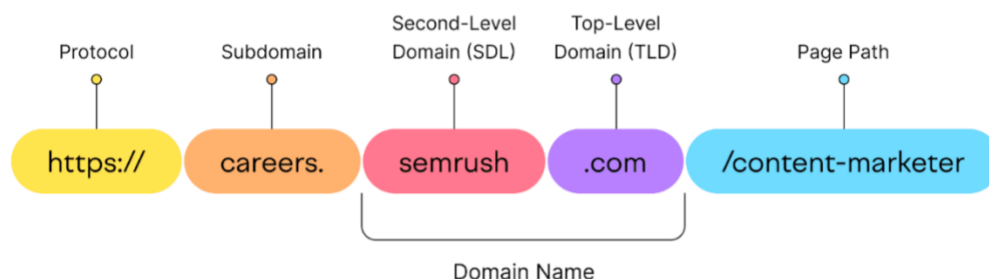Figure 1: Table containing main Internet terminologies

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
   a. 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"
   b. Then it would make a request to the ".com" TLD name server and return a response of the respective authoritative name server

c. 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

d. 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
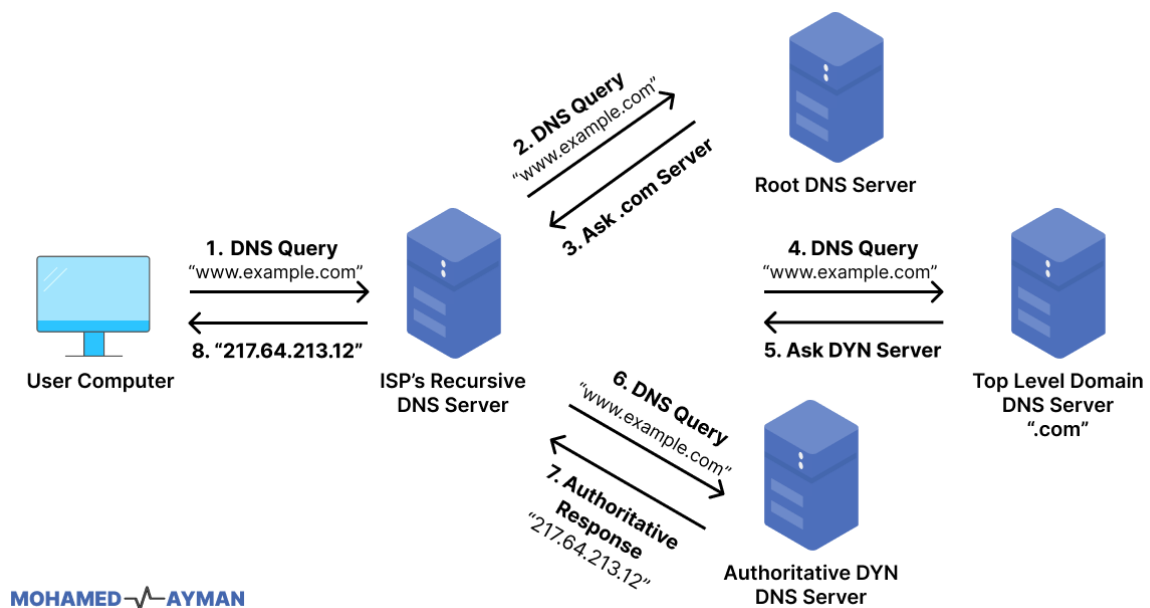

Figure 3: DNS Process

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.
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.
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/.	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.

	c. Create this worker using the Wrangler CLI, upload your Workers code to a public Git repository for your implementation.	
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.

Figure 4: Summary and Breakdown of project steps

Step 1 – Proxy traffic to origin web server through Cloudflare

Step 1 Step-by-Step Guide

1. Created domain name through Cloudflare
2. Created ec2 ubuntu instance on AWS
3. Installing nginx web server in the instance:
   a. SSH into the ubuntu instance first to enter the remove environment, via private pem key file generated during instance creation: `sudo ssh -i <pem.key> ubuntu@public-ip`
   b. Once you have entered the remote environment, you have to first install nginx module via `sudo apt-get install nginx`. This will also automatically start the nginx service, which you can check its status via `sudo systemctl status nginx`
4. Create the flask app script to return the HTTP request headers accordingly (refer to the GitHub repo > 1.cloudflare-proxy > app.py)
5. Install pip packager via `sudo apt install python3-pip`
6. With pip installed, you can now install required python modules to run the flask app: `pip3 install flask`
7. Add an A type DNS record so that www.limyinshan.com can point to the public IP of the web server and have its traffic proxied through Cloudflare, configurations as seen below

Type ▲	Name	Content	Proxy status	TTL
A	💬 www	44.203.2.85	☁ Proxied	Auto
Type	Name (required)	IPv4 address (required)	Proxy status	TTL
A ▼	www	44.203.2.85	🔘 ☁ Proxied	Auto
	Use @ for root			

Figure 5: Assigning Type A DNS record

   a. A type DNS record is being created instead of AAAA record as web server is using IPv4 address
   b. It specifies the destination for the "www" subdomain, allow
8. Modify the nginx.conf file via `sudo nano /etc/nginx/sites-available/default` to listen to port 80 and have it reverse proxy to port 5000 because the flask app is running on port 5000
   a. Ensure that it listens to port 80: `listen 80;`
   b. Ensure that it has reverse proxy configurations by setting `proxy_pass http://127.0.0.1:5000` so that the default HTTP request will be forwarded to the web server and call the endpoint running on port 5000
9. Test the config file syntax using `sudo nginx -t` to ensure that it can work properly and restart the nginx server to update the config file changes using `sudo service nginx restart`
10. Run the flask app: `python3 app.py`
    a. If permission is denied for flask app to run, enable the respective read write permissions first (chmod 707)
    b. Head over to www.limyinshan.com and you should be able to see the request headers as the response body
11. To ensure that the flask app runs even after I exit the ec2 instance, instead of just running the app.py file in step 10, use `nohup python3 app.py &`
    a. This ensures that the app will run in the background even after hang up / logout

> b. You should be able to double check that the process is running using `pgrep python3` and also get its process id
>
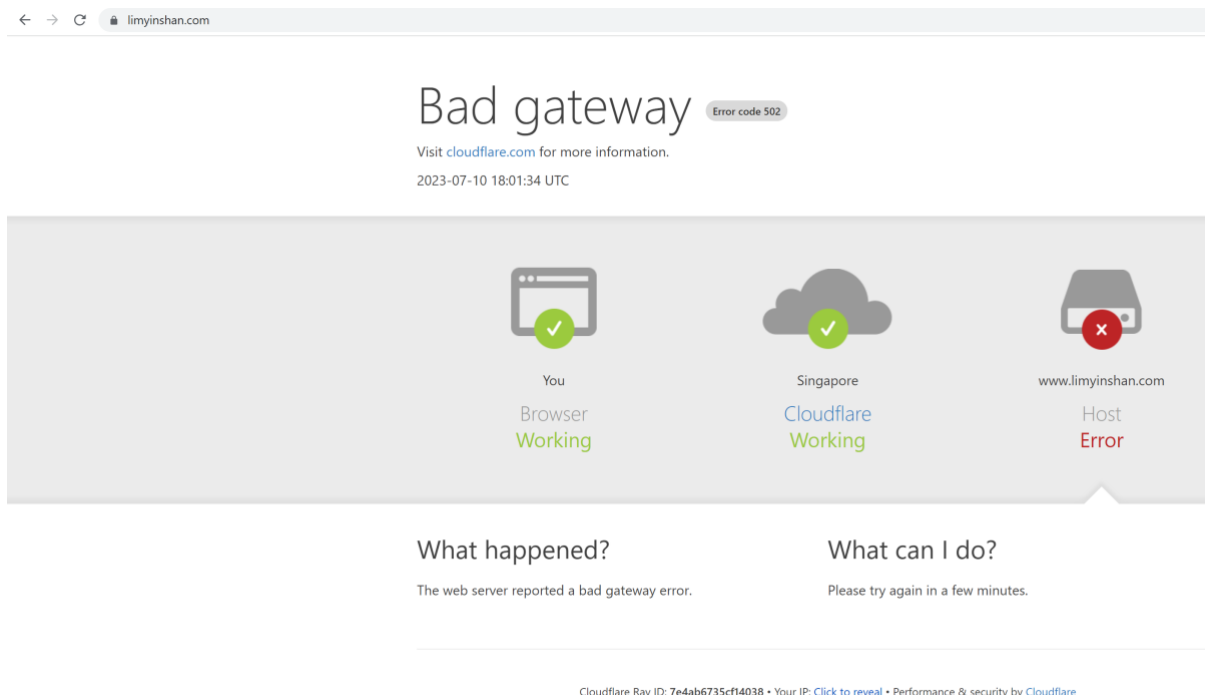> c. You can use `sudo kill <process-id>` to kill the process

Figure 6: Illustrating successful proxy through Cloudflare (by not running the flask app)
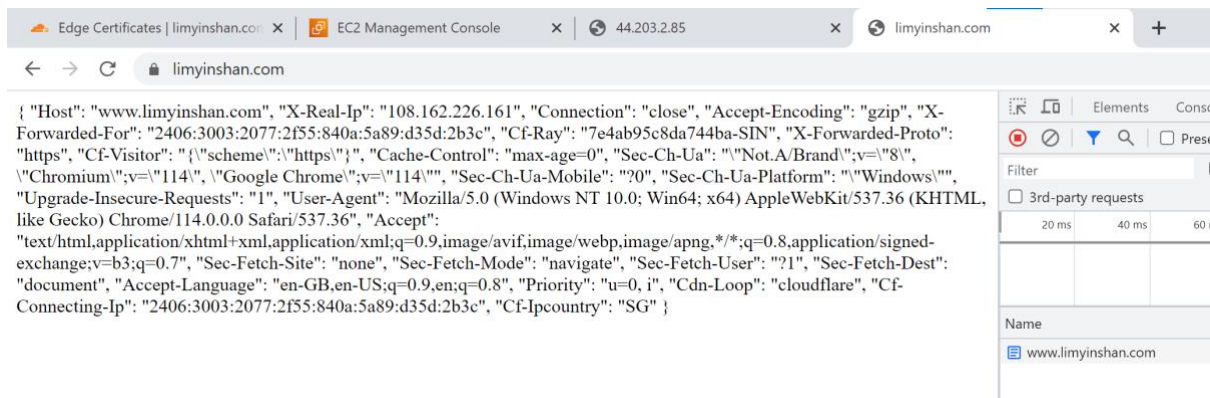

Figure 7: Response body received after running flask app

The use of Step 1 and its benefits

As mentioned earlier, the main purpose for Step 1 is to proxy traffic through Cloudflare, that is, routing incoming web traffic through Cloudflare's network before reaching the origin server (ubuntu instance). This is so that Cloudflare would be the one managing the DNS management instead, such that we can reap its benefits. These are just some of the many benefits from doing so:

1. Traffic distribution and load balancing:
   a. When Cloudflare proxy is enabled for your domain, incoming requests are automatically distributed across Cloudflare's global network of data centres. This

helps distribute the load across and handle traffic spikes more efficiently. Cloudflare's intelligent routing algorithm also directs each request to the nearest data centre, reducing latency and improving response times.

2. DDoS protection and security:
   a. "Distributed Denial of Service (DDoS) is a malicious attempt to disrupt the normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic." This means overflooding the server, causing it to reach its limits and prevent other people from accessing the server, just like a traffic jam.
   b. Cloudflare provides robust protection against these DDoS attacks by filtering and absorbing malicious traffic before it reaches the origin server. This helps ensure the website remains available and accessible during such attacks.
   c. Cloudflare also offers various security features such as web application firewall (WAF), SSL/TLS encryption, IP reputation-based blocking, and bot mitigation, which further enhances the overall security and integrity of your website. (You will see more of this in Step 2)

3. Content caching and optimization:
   a. Cloudflare caches static content from your website on its edge servers, closer to the end-users. This reduces the load on the 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. (You will see more of this in Step 5 when dealing with Cloudflare Workers)

4. SSL/TLS termination:
   a. Cloudflare acts as a SSL/TLS termination point, allowing you to secure your website with HTTPS even if the 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:
   a. Cloudflare provides detailed analytics and insights about the website's traffic, including visitor statistics, bandwidth usage, and security events.
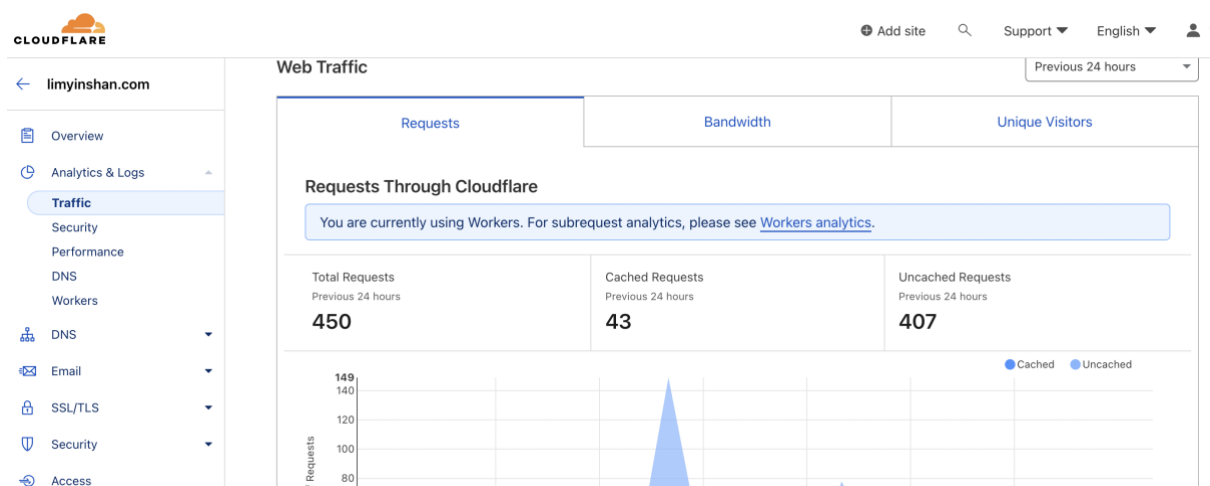

Figure 8: Cloudflare Analytics Dashboard

Step 2 – Securing the communication between Cloudflare and server

Since we have proxied traffic through Cloudflare, we can enjoy encryption of traffic between the browser and Cloudflare. However, the communication between Cloudflare and the server is still not yet secured.
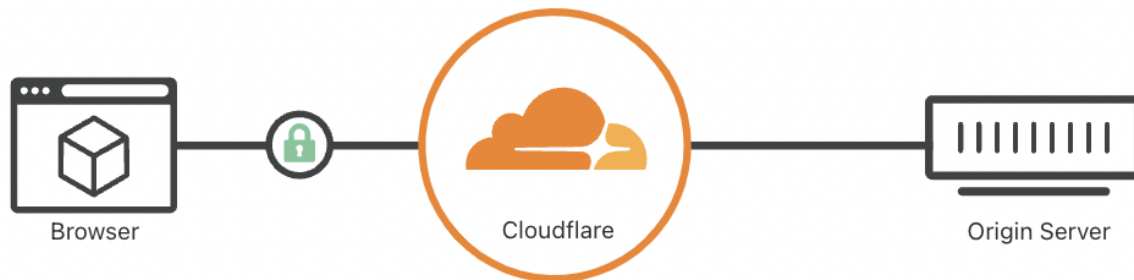

Figure 9: Partial SSL/TLS encryption

TLS Certificate

A TLS certificate is a digital certificate that establishes the identity of a server and enable secure communication using TLS/SSL protocols, which are basically protocols that provide encryption and authentication mechanisms to ensure integrity, authenticity and confidentiality of data transmitted between a client and server. They are validated during the TLS handshake process, which is a series of information exchange steps between the client and the server.

Step 2 Step-by-Step Guide

Step 2 goes about illustrating how to secure the remaining portion of the traffic flow, with a third-party generated TLS certificate.

1. Choosing the certificate authority (CA) – I chose Let's Encrypt as the third-party CA as it was not only fairly easier to obtain the TLS certificate, but also credible, free of charge and has auto-renewal. The TLS certificate can be obtained via certbot
2. Installing snap for certbot installation: `sudo apt install snapd`
3. Install certbot for to generate TLS certificate: `sudo snap install --classic certbot`
4. Run certbot to obtain TLS certificate: `sudo certbot certonly --nginx --preferred-challenges http-01`
   a. The certonly command ensures that certbot only obtains the certificate without modifying the Nginx and Cloudflare configurations (for precaution)
   b. Follow the respective instructions that appear in the terminal for the certification generation, such as the one below:

```
Which names would you like to activate HTTPS for?
We recommend selecting either all domains, or all domains in a VirtualHost/server block.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1: www.limyinshan.com
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Select the appropriate numbers separated by commas and/or spaces, or leave input
blank to select all options shown (Enter 'c' to cancel): 1
Requesting a certificate for www.limyinshan.com

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/www.limyinshan.com/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/www.limyinshan.com/privkey.pem
This certificate expires on 2023-10-08.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.
```

Figure 10: Instructions during TLS certificate generation via certbot

5. Update the nginx config file with the newly generated certificate and restart service
    a. Below is an example of how to include the certificate in the nginx config file:

```
# listen 80;
listen 443 ssl;
# listen [::]:80 default_server;

# SSL configuration
ssl_certificate /etc/letsencrypt/live/www.limyinshan.com/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/www.limyinshan.com/privkey.pem;
```

Figure 11: SSL configuration in Nginx config file

    b. To test syntax: `sudo nginx -t`
    c. Restart the nginx service: `sudo systemctl restart nginx`
6. Certbot has set up automatic renewal for the certificate, so it will be automatically renewed before it expires
7. After certificate is created and activated on the server, enable Full (Strict) SSL/TLS Encryption mode on Cloudflare Dashboard such as the picture below:
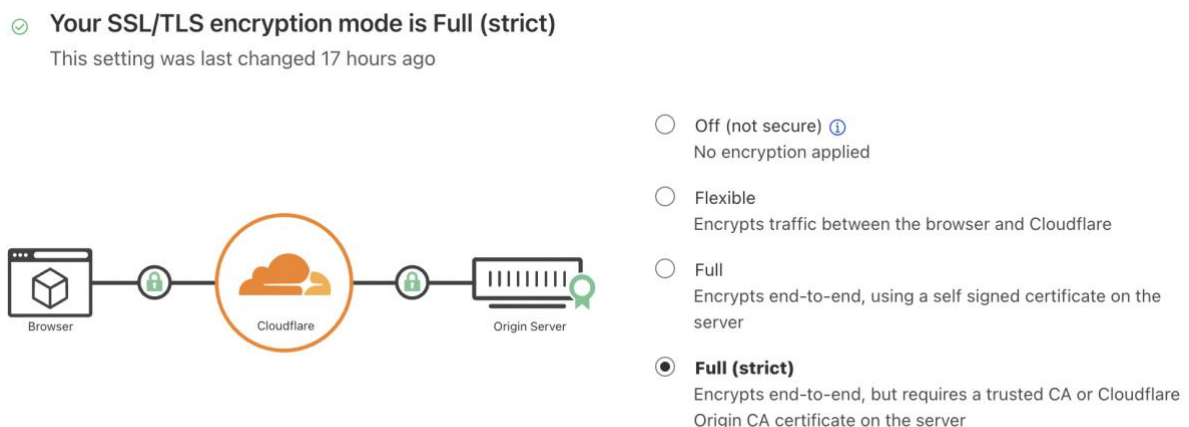

Figure 12: SSL/TLS Encryption mode set to Full (Strict)

Now that you have successfully generated the TLS certificate on your server and activated full encryption mode, you should be able to view the certificate on the web browser by clicking on the "lock" icon beside the URL hostname. It is issued by "Let's Encrypt", the CA that I chose at the start.
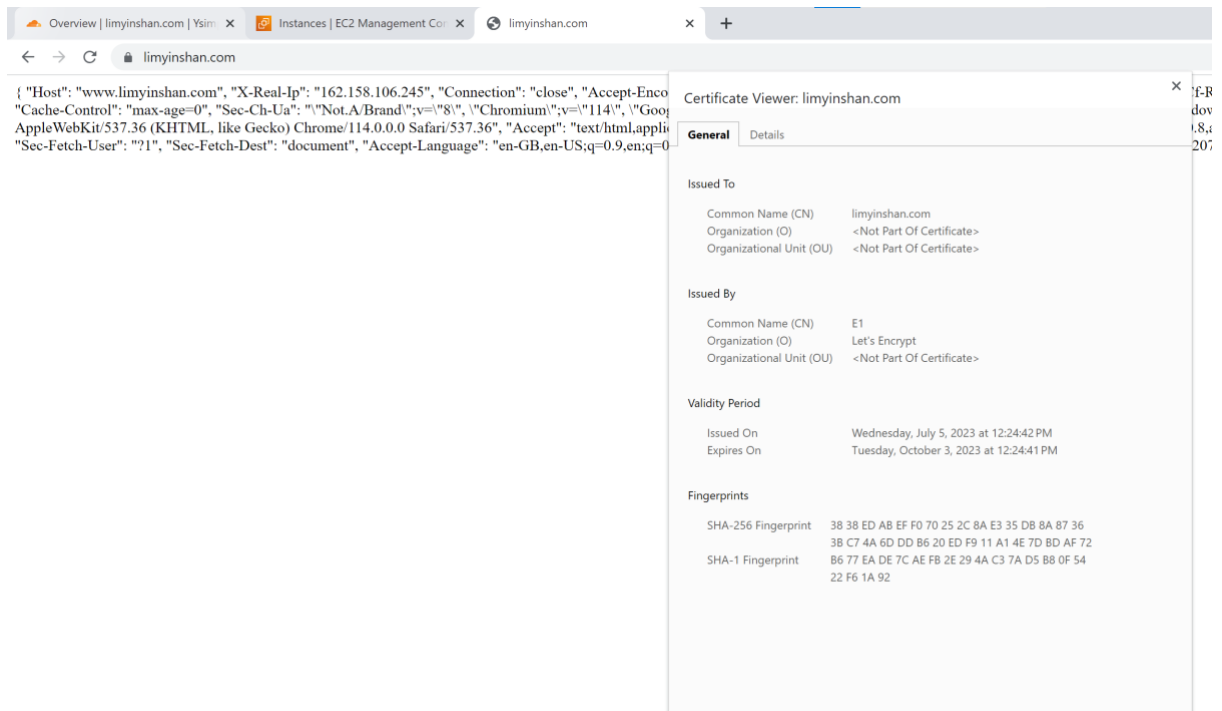
{ "Host": "www.limyinshan.com", "X-Real-Ip": "162.158.106.245", "Connection": "close", "Accept-Enco
"Cache-Control": "max-age=0", "Sec-Ch-Ua": "\"Not.A/Brand\";v=\"8\", \"Chromium\";v=\"114\", \"Goog
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36", "Accept": "text/html,appli
"Sec-Fetch-User": "?1", "Sec-Fetch-Dest": "document", "Accept-Language": "en-GB,en-US;q=0.9,en;q=0

Certificate Viewer: limyinshan.com

General Details

Issued To

Common Name (CN)	limyinshan.com
Organization (O)	<Not Part Of Certificate>
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	E1
Organization (O)	Let's Encrypt
Organizational Unit (OU)	<Not Part Of Certificate>

Validity Period

Issued On	Wednesday, July 5, 2023 at 12:24:42 PM
Expires On	Tuesday, October 3, 2023 at 12:24:41 PM

Fingerprints

SHA-256 Fingerprint	38 38 ED AB EF F0 70 25 2C 8A E3 35 DB 8A 87 36 3B C7 4A 6D DD B6 20 ED F9 11 A1 4E 7D BD AF 72
SHA-1 Fingerprint	B6 77 EA DE 7C AE FB 2E 29 4A C3 7A D5 B8 0F 54 22 F6 1A 92

Figure 13: TLS Certificate Viewer

Step 3 – Secure connections using Cloudflare Tunnel

Cloudflare Tunnel

As mentioned earlier, the purpose of step 3 is to create a way to create a secure connection between your resources that do not have a public IP, and the internet through Cloudflare's network. It creates a secret tunnel between your servers and Cloudflare's edge network, ensuring that all traffic between the two is encrypted and protected. Tunnel is typically used to easily yet securely expose private origin servers to the internet whilst benefitting from Cloudflare's security and performance features.

Figure 14: A clear process view of how Cloudflare Tunnel enables communication

Step 3 Step-by-Step Guide

This is how to set up the Cloudflare Tunnel.

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 in the terminal like below:

```
ubuntu@ip-172-31-93-213:~$ cloudflared tunnel login
Please open the following URL and log in with your Cloudflare account:

https://dash.cloudflare.com/argotunnel?aud=&callback=https%3A%2F%2Flogin.cloudf
M06rBCR6184-PC-NqFE%3D

Leave cloudflared running to download the cert automatically.
You have successfully logged in.
If you wish to copy your credentials to a server, they have been saved to:
/home/ubuntu/.cloudflared/cert.pem
```

3. Create a cloudflare tunnel: `cloudflared tunnel create limyinshan-tunnel`
   a. "limyinshan-tunnel" being the name of the tunnel
4. Verify that it is created using: `cloudflared tunnel list`

```
ubuntu@ip-172-31-93-213:~$ cloudflared tunnel list
You can obtain more detailed information for each tunnel with `cloudflared tunnel info <name/uuid>`
ID                                   NAME              CREATED                CONNECTIONS
80f7bf30-becc-4e86-84b4-c984a7c9da6a limyinshan-tunnel 2023-07-10T19:15:41Z
```

Figure 16: Verifying Cloudflare Tunnel creation

5. Create a config file to route traffic from the origin server to the hostname tunnel as per below:

```
ubuntu@ip-172-31-93-213:~/.cloudflared$ cat config.yml
tunnel: 80f7bf30-becc-4e86-84b4-c984a7c9da6a
credentials-file: 80f7bf30-becc-4e86-84b4-c984a7c9da6a.json
url: http://localhost:5000
```

Figure 17: Cloudflare Tunnel config file details

   a. Note that it is pointing to the origin server directly, hence the url is localhost:5000
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. Now that the tunnel is working and to run it permanently:
   a. Create a shell script that enters the cloudflare working directory and run the commands, like the one below

```
ubuntu@ip-172-31-93-213:~$ cat cloudflared-start.sh
#!/bin/bash

cd ~/.cloudflared
nohup cloudflared tunnel run limyinshan-tunnel > cloudflared.log 2>&1 &
```

Figure 18: Shell script details that runs the Cloudflare Tunnel in the background

   b. You can view the shell script in the GitHub repo > 3.cloudflare-tunnel > cloudflared-start.sh

Step 4 – Cloudflare API

Step 4 Step-by-Step Guide

1. Create an API scoped token just for retrieving the DNS records
   a. Cloudflare dashboard > My profile > API Tokens > Create token
   b. Use "Edit Zone DNS" template
   c. Allow Read permissions
   d. 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 like the picture below

API Tokens

Edit zone DNS API token was successfully created

Copy this token to access the Cloudflare API. For security this will not be shown again. learn more

PZB7ETR1xyQAlDDIMwaWT2ISxjCk7Hq-ZeiTr7f1   Copy

Figure 19: Cloudflare API Scoped Token generated

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

- You can view the json file in the GitHub repo > 4.cloudflare-api > dns-records.json

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
	  o Get the respective zone_id by **excluding** the paths after "zones" in the function call
	  o 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 – Cloudflare Workers, Wrangler CLI

The shift from server-computing to edge-computing and Cloudflare Workers

Traditionally, web applications run on origin servers where requests are processed and responses are generated. However, as the world becomes increasingly global and distributed, this traditional approach can pose latency and performance issues in the long run. This brought rise to a service known as Cloudflare Workers, who can essentially move the code execution from backend origin servers to Cloudflare's edge locations globally. Because Workers can easily deploy and scale code logic execution closer to the user, this enables much faster response times, improved performance, reduced latency and ultimately also reducing the load on the origin server.

With Cloudflare Workers, it provides a serverless compute platform at the edge of Cloudflare's network, increasing personalisation and speed.

Step 5 Step-by-Step Guide

Step 5 illustrates the creation and deploying of a Cloudflare worker.

1. Part of the requirement is to use the Wrangler CLI, hence, Install wrangler via `npm install wrangler -g` (this assumes you already have both npm and node installed, and that the node version is of 16.13.0 or later)
2. Create Cloudflare worker via CLI: `wrangler init cloudflare-worker-proj` and follow the instructions accordingly
   a. Choose the option "Hello World" Worker as you only need a simple base to implement the required functionality
   b. Do not deploy yet as there are still some configurations to complete
3. You can refer to the GitHub > 5.cloudflare-worker for the scripts to create the worker
   a. Update the src/worker.js file to fetch the required response body
   b. Update wrangler.toml file to set instructions for worker creation
4. Deploy worker to cloudflare using: `wrangler deploy`
   a. Very fast in updating the js file, changes are reflected on "/geo" route really quickly
5. After creating the worker via Wrangler CLI, create and configure the route for "/geo" path in the Cloudflare Dashboard, such as the picture below

Figure 20: Configuration of Worker Route for /geo path

You should be able to visit the "/geo" path and see the reflected changes as below.

This is your 2406:3003:2077:2f55:840a:5a89:d35d:2b3c and you are accessing this site from SG | 55430
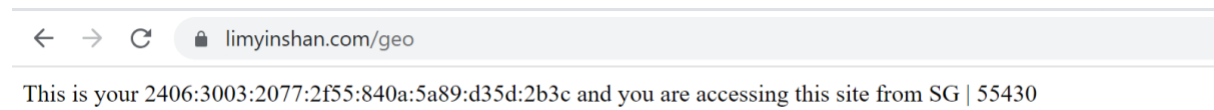
Figure 21: /geo path response body

Step 6 – Cloudflare Zero Trust System

As mentioned earlier, this last step is to create an easy way to only allow restricted and authorised users to access certain paths or subdomains, based on certain conditions and policies set. In particular, email is being used as the single sign-on parameter and authentication based on tokens and sessions.

Step 6 Step-by-Step Guide

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:
   a. session duration as "no duration, expires immediately" for debugging purposes
   b. subdomain as tunnel
   c. domain as limyinshan.com
   d. path as geo
   e. Ensure "Enable App in App Launcher" is checked and using default domain
   f. 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)

Figure 22: allow-me policy to only allow my email

block-all-emails

Figure 23: block-all-emails policy to literally block all emails except my email

    i. This is ensure that only my email ysimptstuff@gmail.com is able to access https://tunnel.limyinshan.com/geo.

    ii. One thing to note would be that if you were to key in an email that is not the allowed email list, 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
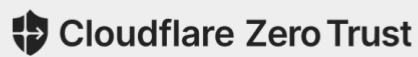
Figure 24: Cloudflare Access – Email Sign-in
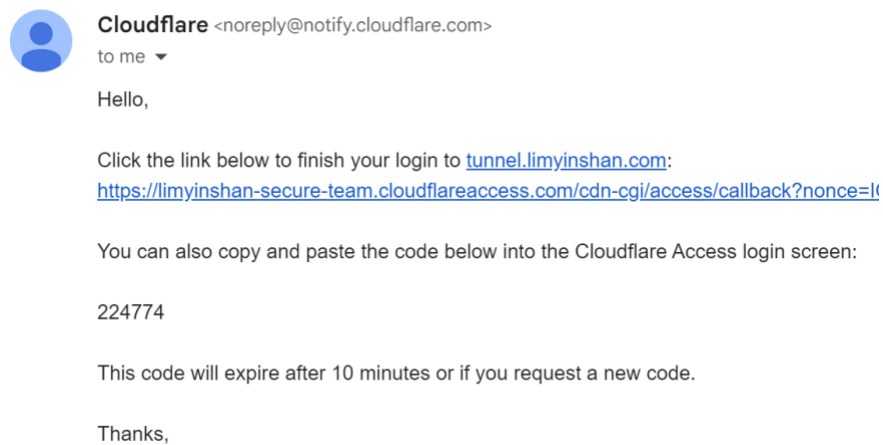
Figure 25: Code being sent to email

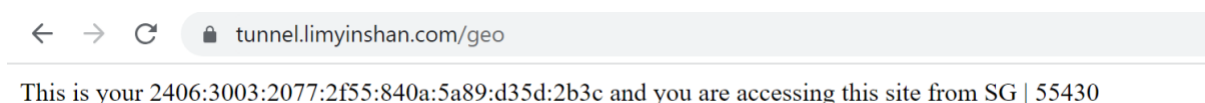Figure 26: One time code sent via email

Figure 27: Access granted only after all conditions met

Preventing Cloudflare bypass

5. 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 Cloudflare IP ranges, on both Cloudflare and AWS servers.

	Name	Security group rule... ▽	IP version ▽	Type ▽	Protocol ▽	Port range ▽	Source ▽	Description
☐	–	sgr-0fb7fe19a310e5d09	IPv4	HTTPS	TCP	443	131.0.72.0/22	–
☐	–	sgr-0a4b657956e370a...	IPv4	HTTPS	TCP	443	198.41.128.0/17	–
☐	–	sgr-00f06cf797646868a	IPv4	HTTPS	TCP	443	173.245.48.0/20	–
☐	–	sgr-08505683fed796fba	IPv4	HTTPS	TCP	443	188.114.96.0/20	–
☐	–	sgr-085c98d20736b47f8	IPv4	HTTPS	TCP	443	197.234.240.0/22	–
☐	–	sgr-01ab929bba6a33d...	IPv4	HTTPS	TCP	443	103.21.244.0/22	–
☐	–	sgr-06aff71f27097e906	IPv4	HTTPS	TCP	443	108.162.192.0/18	–
☐	–	sgr-0628c209f5fa9425c	IPv4	HTTPS	TCP	443	141.101.64.0/18	–
☐	–	sgr-0a66140a04a68d6...	IPv4	HTTPS	TCP	443	162.158.0.0/15	–
☐	–	sgr-02628a361e47efe8d	IPv4	HTTPS	TCP	443	103.22.200.0/22	–
☐	–	sgr-0d7e01cd59ba1ef04	IPv4	HTTPS	TCP	443	172.64.0.0/13	–
☐	–	sgr-0b409b589b3adee...	IPv4	HTTPS	TCP	443	104.16.0.0/13	–
☐	–	sgr-0cad6dedc4b28855b	IPv4	HTTPS	TCP	443	104.24.0.0/14	–
☐	–	sgr-060052ff6498bd076	IPv4	HTTPS	TCP	443	103.31.4.0/22	–
☐	–	sgr-07748a7c3885b05...	IPv4	HTTPS	TCP	443	190.93.240.0/20	–

Figure 28: Inbound rules set to only accept traffic from Cloudflare's range of Ips

Gateway / Firewall Policies

Firewall policies

Protect your users by creating policies that scan, filter, and log traffic. By default, Gateway allows all traffic and DNS queries unless a policy matches. 📖 Learn more

DNS	Network	HTTP

Network policies

+ Add a policy

#	Policy name	Policy ID	Action	Enabled
1	allow-cloudflare-ips	a5304b9a-49bc-49f5-97a8-b93d7ffc504c	ALLOW	✓
2	block-non-cloudflare-ips	aa940f1a-bad0-4339-ae96-ae086911f1ac	BLOCK	✓

Figure 29: Firewall policies created to only accept traffic from Cloudflare's range of IPs

Challenges through the project

As I did not have much hands-on experience and knowledge on both the Internet architecture and Cloudflare's product and services, a lot of time was invested in the research phase – from watching YouTube videos, reading Cloudflare's documentations to seeking help from stackoverflow and ChatGPT.

The next half would be the implementation phase itself, which I believed was both nerve-wrecking and fun. Even though there was a lot of debugging and error resolution, I found the process to be really fun and fruitful, as debugging would inherently rely on my comprehension of the product and concept, and successful debugging would mean I knew where my problems lie and how to fix them.

While the project was mostly to test my understanding and application of Cloudflare's technologies, I found the assignment to be both excruciating yet exciting when I finally understood certain concepts I could not previously. It was definitely an amazing eye-opening experience as well as a feeling of accomplishment when I was able to see my website running on the Internet safely and securely, as if I had finally found a place in this huge WWW.

I do hope that this guide was written clearly and precisely such that even an unexperienced and non-technical person may easily understand and pick up the Cloudflare technology!

Future Improvements

1. Improvements **on server side**
   a. Elastic IP – so that if the server were to go down, the public IP would not change and I do not have to reassociate the domain to the new public IP
   b. Autoscaling servers tied to ELB – so that if a server were to go down, it would not affect the uptime as there would be other servers still running, and new ones scaling up quickly to replace the faulty one